

Space vector data path.

Patent Number: ☐ EP0681236, B1
Publication date: 1995-11-08
Inventor(s): EARLE JOHN (US); GAREY KENNETH E (US); WATSON GEORGE A (US);
BINDLOSS KEITH M (US)
Applicant(s):: ROCKWELL INTERNATIONAL CORP (US)
Requested Patent: ☐ JP8050575
Application
Number: EP19950106843 19950505
Priority Number
(s): US19940238558 19940505
IPC Classification: G06F9/302
EC Classification: G06F9/38T, G06F15/78V, G06F9/302, G06F9/318
Equivalents:

Abstract

A space vector path for integrating SIMD scheme into a general-purpose programmable processor is disclosed. The programmable processor comprises mode means coupled to an instruction means (130, 140) for specifying for each instruction whether an operand is processed in one of vector and scalar modes, processing unit (110) coupled to the mode means for receiving the operand and, responsive to an instruction as specified by the mode means, for processing the operand in one of the vector and scalar modes, wherein the vector mode indicating to the processing unit (110) that there are a plurality of elements within the operand and the scalar mode indicating to the processing unit (110) that there is

one element within the operand. 

Data supplied from the esp@cenet database - I2

(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平8-50575

(43)公開日 平成8年(1996)2月20日

(51)Int.Cl.⁶

G 0 6 F 15/80
9/38

識別記号

庁内整理番号

F I

技術表示箇所

3 1 0 G

審査請求 未請求 請求項の数30 O L (全 34 頁)

(21)出願番号 特願平7-109642

(22)出願日 平成7年(1995)5月8日

(31)優先権主張番号 2 3 8 5 5 8

(32)優先日 1994年5月5日

(33)優先権主張国 米国 (US)

(71)出願人 590002448

ロックウェル・インターナショナル・コー
ポレイション

ROCKWELL INTERNATIO
NAL CORPORATION

アメリカ合衆国、90740-8250 カリフォ
ルニア州、シール・ビーチ、シールビー
チ・プールバード、2201

(72)発明者 ケイス・エム・ビンドロス

アメリカ合衆国、92714 カリフォルニア
州、アーバイン、ビーバー・ストリート、
3861

(74)代理人 弁理士 深見 久郎 (外3名)

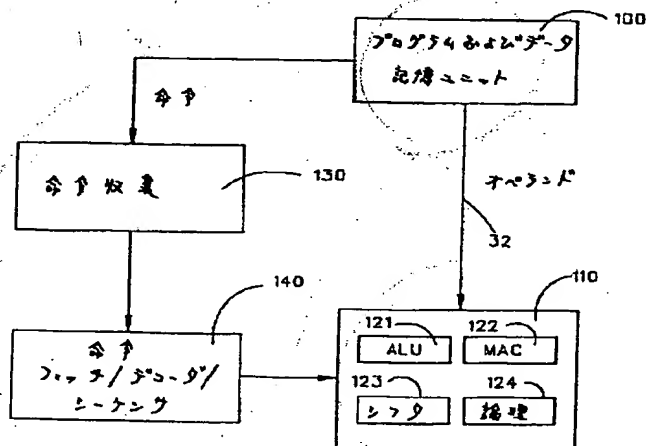
最終頁に続く

(54)【発明の名称】 プログラマブルプロセッサ、前記プログラマブルプロセッサを用いてデジタル信号処理を行なうための方法およびその改良

(57)【要約】

【目的】 汎用プログラマブルプロセッサにSIMDスキームを統合する空間ベクトルデータ経路を提供する。

【構成】 プログラマブルプロセッサは、命令手段に結合され、オペランドがベクトルおよびスカラーモードの1つ処理されるか否かを各命令に対して特定するためのモード手段と、モード手段に結合され、オペランドを受取り、モード手段に特定されるような命令に応答してベクトルおよびスカラーモードの1つでオペランドを処理するための処理ユニット110を含み、ベクトルモードはオペランド内に複数の要素があることを処理ユニット110に示し、スカラーモードはオペランド内に1つの要素があることを処理ユニット110に示す。



【特許請求の範囲】

【請求項 1】 少なくとも 1 つのオペランドの複数データ経路処理のためのプログラマブルプロセッサであって、各オペランドは少なくとも 1 つの要素を含み、前記プロセッサは命令フェッチ／デコード／シーケンス手段によって判断される予め定められるシーケンスで命令を実行し、前記プログラマブルプロセッサは、

- a) 前記命令手段に結合され、前記少なくとも 1 つのオペランドがベクトルおよびスカラモードのうちの 1 つで処理されるかどうかを各命令に対して特定するためのモード手段と、
- b) 前記モード手段に結合される処理ユニットとを含み、前記処理ユニットは、前記少なくとも 1 つのオペランドを受取り、前記モード手段によって特定される前記命令にตอบสนองして前記ベクトルおよびスカラモードのうちの 1 つにある前記少なくとも 1 つのオペランドを処理し、前記ベクトルモードは複数の要素が前記オペランド内にあることを前記処理ユニットに示し、前記スカラモードは 1 つの要素が前記オペランド内にあることを前記処理ユニットに示す、プログラマブルプロセッサ。

【請求項 2】 前記処理ユニットは、

- a) 前記モード手段からの命令にตอบสนองし、前記少なくとも 1 つのオペランドにある各それぞれの要素を同時並列処理して、前記ベクトルモードの各それぞれの要素に対する独立した結果を得るための第 1 のベクトル手段と、
- b) 前記モード手段からの前記命令にตอบสนองし、前記少なくとも 1 つのオペランドにある第 1 の要素を、前記ベクトルモードの前記オペランドにある少なくともひとつの第 2 の要素との選択的組合せで処理するための第 2 のベクトル手段と、
- c) 前記モード手段からの前記命令にตอบสนองし、前記オペランドの各それぞれの部分を処理してそれぞれの部分的結果を得、各それぞれの部分的結果を併せて前記スカラモードでスカラ結果を引出すためのスカラ手段とを含む、請求項 1 に記載のプログラマブルプロセッサ。

【請求項 3】 前記第 1 のベクトル手段およびスカラ手段は、

- a) 複数の乗算累算器と、
- b) 複数のシフタと、
- c) 複数の算術ユニットと、
- d) 論理ユニットとのうちの少なくとも 1 つを備え、各々は、ベクトルオペランド内の少なくともひとつのそれぞれの要素と、スカラオペランドのそれぞれの部分とのうちの 1 つを処理する、請求項 2 に記載のプログラマブルプロセッサ。

【請求項 4】 前記スカラ手段は条件付移動を行ない、前記第 2 のベクトル手段と前記スカラ手段とは前記第 2 のベクトルモードにある前記オペランド内の前記第 1 および第 2 の要素の前記選択的組合せに基づいて条件付き

分岐を行なう、請求項 3 に記載のプログラマブルプロセッサ。

【請求項 5】 前記処理ユニットは、

- a) 前記ベクトルおよびスカラモードのうちの 1 つで動作する複数の加算器を備え、前記複数の加算器の各加算器は前記モード手段によって特定されるベクトルオペランドからの要素を受取りそれを個別に処理し、前記複数の加算器は前記モード手段によって特定されるスカラオペランドを受取ってそれをともに処理し、前記処理ユニットはさらに、
- b) 前記複数の加算器に結合され、前記複数の加算器が 1 つの加算器として前記スカラオペランドを処理するよう、キャリースステータスを前記スカラモードで前記複数の加算器の各々の間で送るための加算器制御手段を備える、請求項 1 に記載のプログラマブルプロセッサ。

【請求項 6】 前記複数の加算器は 1 つの加算器として前記スカラオペランドを処理するよう、前記加算器制御手段は前記スカラモードで前記複数の加算器の各々の間でオーバフローステータスをさらに送る、請求項 5 に記載のプログラマブルプロセッサ。

【請求項 7】 前記処理ユニットは、

- a) 前記ベクトルおよびスカラモードのうちの 1 つで動作する複数の乗算累算器 (MAC) を備え、各 MAC は前記モード手段によって特定されるようなベクトルオペランド内の要素を受取ってそれを別個に同時並列処理し、前記複数の MAC は前記モード手段によって特定されるスカラオペランドを受取りそれをともに処理し、前記処理ユニットはさらに、
- b) 前記 MAC に結合され、前記モード手段にตอบสนองし、前記複数の MAC を、ベクトルモードでは互いに独立して動作させ、スカラモードではともに動作させるための MAC 制御手段を備える、請求項 1 に記載のプログラマブルプロセッサ。

【請求項 8】 前記処理ユニットは、

- a) 前記ベクトルおよびスカラモードのうちの 1 つで動作する複数の論理ユニットを備え、前記複数の論理ユニットの各論理ユニットは前記モード手段によって特定されるようなベクトルオペランド内の要素を受取ってそれを別個に同時並列処理し、前記複数の論理ユニットは前記モード手段によって特定されるスカラオペランドを受取ってそれをともに処理し、前記処理ユニットはさらに、
- b) 前記複数の論理ユニットに結合され、前記複数の論理ユニットが 1 つの論理ユニットとして前記スカラオペランドを処理するよう、前記スカラモードで前記複数の論理ユニットの各々の間でゼロステータスを送るための、論理制御手段を備える、請求項 1 に記載のプログラマブルプロセッサ。

【請求項 9】 前記処理ユニットは、

- a) 前記スカラモードでは 1 つの統合化されたシフタ

として、および前記ベクトルモードでは複数のシフトとして、選択的に動作するための複数のシフトを備え、前記複数のシフトの各々は、演算の第1のモードにある前記モード手段にตอบสนองし、特定されるベクトルオペランドからの要素を受取ってそれを別個に同時並列処理し、前記複数のシフトは、第2のモードの演算にある前記モード手段にตอบสนองし、スカラオペランドを受取ってそれとともに処理し、前記処理ユニットはさらに、

b) 前記シフトに結合され、前記複数のシフトが前記スカラオペランドを処理するよう、シフトされたオペランドビットを前記スカラモードで前記複数のシフトの各々の間で送るためのシフト制御手段を含み、前記シフト制御手段は、前記ベクトルモードでは、前記複数のシフトの各々からシフトされたオペランドビットを送ることを不能化する、請求項1に記載のプログラマブルプロセッサ。

【請求項10】 スカラおよびベクトルモードの1つでオペランドの条件を評価するための比較手段をさらに含む、請求項1に記載のプログラマブルプロセッサ。

【請求項11】 前記比較手段は命令実行のシーケンスを修飾するために各オペランドの条件を評価する、請求項10に記載のプログラマブルプロセッサ。

【請求項12】 第1のオペランドは前記比較手段に基づいて第1の記憶位置から第2の記憶位置へ条件付きで移動され、前記比較手段は、第1のオペランド内の対応する要素が移動されるかどうかを判断するために第2および第3のオペランド内の対応する要素を各々が比較する複数のサブ比較器を含む、請求項10に記載のプログラマブルプロセッサ。

【請求項13】 前記モード手段は、各々の命令が命令単位でベクトルおよびスカラモードのうちの1つを特定するよう、各命令内にフィールドとして含まれる、請求項1に記載のプログラマブルプロセッサ。

【請求項14】 前記モード手段は各命令内にビットフィールドとして含まれる、請求項13に記載のプログラマブルプロセッサ。

【請求項15】 各オペランド内に少なくとも1つの要素を有するオペランドをストアするためのデータメモリと、実行のための命令をストアするための命令メモリと、命令手段と、複数の算術論理ユニット（ALU）とに結合される汎用コンピュータにおける、複数データデジタル信号処理を行なうための構成であって、

a) 前記命令メモリと前記命令手段とに結合され、オペランドが前記処理ユニットによってベクトルモードおよびスカラモードのうちの1つとして処理されるかどうかを各命令において特定するためのモード手段と、

b) 前記モード手段にตอบสนองし、前記ALUを、スカラオペランドの場合は第1のモードで1つのユニットとしてともに動作させ、ベクトルオペランドの場合には各ユニットが第2のモードにある状態で個々の演算ユニット

として独立して動作させることを選択的に行なうためのALU制御手段と、

c) 前記ALU制御手段と前記ALUとに結合され、スカラオペランドの場合には前記ALUの各々の間でキャリー条件を選択的に送り、ベクトルオペランドの場合には前記ALUの各々のための前記キャリー条件を無視するための、キャリー条件手段とを備える、各オペランド内に少なくとも1つの要素を有するオペランドをストアするためのデータメモリと、実行のための命令をストアするための命令メモリと、命令手段と、複数の演算論理ユニット（ALU）とに結合される汎用コンピュータにおける、複数データデジタル信号処理を行なうための構成。

【請求項16】 各々がその中に少なくとも1つの要素を有するオペランドをストアするためのデータメモリと、実行のための命令をストアするための命令メモリと、命令手段と、第1の乗算累算器（MAC）とに結合される汎用コンピュータにおける、複数データデジタル信号処理を行なうための構成であって、

a) 前記命令メモリと前記命令手段とに結合され、オペランドが前記処理ユニットによってベクトルモードおよびスカラモードのうちの1つとして処理されるかどうかを各命令において特定するためのモード手段と、

b) 複数のMACと、

c) 前記第1および複数のMACの各々に結合され、前記モード手段にตอบสนองし、前記第1および複数のMACの各々を、ベクトルモードでは互いに独立して動作させ、スカラモードではともに動作させることを選択的に行なうためのMAC制御手段とを備える、複数データデジタル信号処理を行なうための構成。

【請求項17】 ALUによるオペランドの処理は、

a) 前記オペランド内の各独立した要素に結合される各条件コードの組と、

b) 前記オペランド内の組合せにおける選択的組合せの、条件コードの複数の組と、

c) 前記スカラオペランドのための条件コードの前記1つの組とのうちの1つに基づいて修飾される、請求項15に記載の構成。

【請求項18】 前記命令の実行のシーケンスは、

a) 前記オペランド内の各独立した要素に関連する各条件コードの組と、

b) 選択的組合せにある条件コードの複数の組とのうちの1つによって、第1の命令から第2の命令に修飾される、請求項15に記載の構成。

【請求項19】 オペランドは、

a) 前記オペランド内の各々の独立した要素と関連する各条件コードの組と、

b) 選択的組合せにある条件コードの複数の組と、

c) 前記スカラオペランドのための条件コードの前記1つの組とのうちの1つに基づいて、第1の記憶位置か

ら第2の記憶位置へ選択的に移動される、請求項18に記載の信号プロセッサ。

【請求項20】 a) 前記スカラモードでは1つの統合されたシフトとして、および前記ベクトルモードでは複数のシフトとして、選択的に動作するための複数のシフトをさらに含み、前記複数のシフトの各々は、演算の第1のモードにある前記モード手段に回答し、特定されるようなベクトルオペランドからの要素を受取ってそれを独立して処理し、前記複数のシフトは、第2のモード演算にある前記モード手段に回答し、スカラオペランドを受取ってそれとともに処理し、さらに、

b) 前記シフトに結合され、前記複数のシフトが前記スカラオペランドを処理するよう、シフトされたオペランドビットを前記スカラモードで前記複数のシフトの各々の間で送るためのシフト制御手段を含み、前記シフト制御手段は前記ベクトルモードでは前記複数のシフトの各々からシフトされたオペランドビットを送ることを不能化する、請求項15に記載の構成。

【請求項21】 汎用コンピュータを用いる複数データ経路計算のためのプログラマブルプロセッサであって、前記汎用コンピュータは、オペランドをストアするためのデータメモリと、データメモリからオペランドを転送するためのメモリアクセスバスと、実行のための命令をストアするための命令メモリと、前記命令メモリに結合され前記命令のフェッチ、デコードおよび順序付けのための命令手段とを含み、前記プログラマブルプロセッサは、

a) 前記命令手段に結合され、データメモリからのオペランドが単一データ経路モードおよび複数データ経路モードのうちの1つで処理されるかどうかを各命令において特定するためのモード手段を含み、

b) 各データ経路は、演算ユニットと、

乗算累算器(MAC)とを含み、前記プログラマブルプロセッサはさらに、

c) 前記モード手段に回答し、各データ経路にある前記算術ユニットを、スカラオペランドの場合は1つのモードで1つのユニットとしてともに動作させ、ベクトルオペランドの場合は各ユニットが別のモードにある状態で個々の算術ユニットとして独立して動作させることを選択的に行なうための演算制御手段と、

d) 前記算術制御手段と各経路にある前記算術ユニットとに結合され、スカラオペランドの場合には前記算術ユニットの各々の間でキャリー条件を選択的に送り、ベクトルオペランドの場合には各算術ユニットに対応する前記キャリー条件を不能化するためのキャリー条件手段と、

e) 各MACに結合され、前記モード手段に回答し、ベクトルモードでは各MACを互いから独立させて動作させ、スカラモードではともに動作させることを選択的

に行なうためのMAC制御手段とを含む、プログラマブルプロセッサ。

【請求項22】 a) 前記スカラモードでは1つの統合されたシフトとして、および前記ベクトルモードでは複数のシフトとして、選択的に動作するための複数のシフトをさらに含み、前記複数のシフトの各々は、演算の第1のモードにある前記モード手段に回答し、特定されるベクトルオペランドからの要素を受取ってそれを独立して処理し、前記複数のシフトは、第2のモード演算にある前記モード手段に回答し、スカラオペランドを受取ってそれとともに処理し、さらに、

b) 前記シフトに結合され、前記複数のシフトが前記スカラオペランドを処理するよう、シフトされたオペランドビットを前記スカラモードで前記複数のシフトの各々の間で送るためのシフト制御手段を含み、前記シフト制御手段は前記ベクトルモードでは前記複数のシフトの各々からシフトされたオペランドビットを送ることを不能化する、請求項21に記載のプログラマブルプロセッサ。

【請求項23】 プログラマブルプロセッサを用いて複数データ経路を介してデジタル信号処理を行なう方法であって、前記プログラマブルプロセッサは、少なくとも1つの要素を各々が有する少なくとも1つのオペランドで動作し、前記プログラマブルプロセッサは複数のサブ処理ユニットを有し、前記方法は、

a) 前記プログラマブルプロセッサによって実行されるべき命令の予め定められるシーケンスの中から命令を供給するステップと、

b) 前記命令が、前記プログラマブルプロセッサによる少なくとも1つのオペランド上での処理のスカラおよびベクトルモードのうちの1つを生じさせるステップとを含み、前記スカラモードは前記少なくとも1つのオペランド内に1つの要素があることを前記プログラマブルプロセッサに示し、前記ベクトルモードは前記少なくとも1つのオペランド内に複数のサブ要素があることを前記プログラマブルプロセッサに示し、前記方法はさらに、

c) スカラモードの場合には、前記プログラマブルプロセッサの各サブ処理ユニットは、前記命令に回答し、前記オペランドのそれぞれの部分を受取って処理して部分的小および中間結果を発生するステップと、

d) 各サブ処理ユニットは、その中間結果を前記複数のサブ処理ユニットの間で送り、その部分的結果を他のサブ処理ユニットと併せて前記オペランドのための最終結果を発生するステップと、

e) 第1の条件コードを発生して前記最終結果に対応するステップと、

f) ベクトルモードの場合には、前記プログラマブルプロセッサの各サブ処理ユニットは、前記命令に回答し、前記オペランド内の前記複数のサブ要素からそれぞれ

れのサブ要素を受取ってそれを処理して、各中間結果は不能化されかつ各部分的結果はその対応する要素のための最終結果を表わす状態で部分的および中間結果を発生するステップと、

g) 複数の第2の条件コードを、その各々が独立した結果に対応する状態で発生するステップとを含む、デジタル信号処理方法。

【請求項24】 汎用コンピュータを介する複数データ経路計算のためのプログラマブルプロセッサであって、前記汎用コンピュータは、オペランドをストアするためのデータメモリと、プログラム命令をストアするための命令メモリと、命令手段とを含み、前記プログラマブルプロセッサは、前記命令手段に結合され、データメモリからのオペランドがベクトルおよびスカラモードのうちの1つとして処理されるかどうかを特定するためのモード手段を含み、ベクトルモードは各オペランド内の複数の要素を判断し、スカラモードはオペランド内の1つの要素を判断し、前記プログラマブルプロセッサはさらに、モード手段とデータメモリとに結合される複数の処理ユニットを含み、各処理ユニットはオペランドのそれぞれの要素を受取りそれを処理して、部分的結果および伝搬情報を得るために処理し、前記プログラマブルプロセッサはさらに、前記ベクトルモードで動作し、前記処理ユニットに結合され、各部分的結果を各要素の処理のその最終結果として送り、伝搬情報を無視するためのベクトル手段と、前記スカラモードで動作し、前記処理ユニットに結合され、各部分的結果と伝搬情報とを併せて各オペランドの処理のその最終結果を得るためのスカラ手段とを含む、プログラマブルプロセッサ。

【請求項25】 各処理ユニットは処理条件を保存するための条件コードの組を備え、条件コードの前記組は、
a) 個々に第1のベクトルモードにある各組と、
b) 第2のベクトルモードにある別の組との選択的組合せにある各組と、
c) 前記スカラモードで組合せられるスカラオペランドのすべての組とのうちの1つによって、プログラマブルプロセッサの処理を修飾する、請求項24に記載のプロセッサ。

【請求項26】 各処理ユニットは、
a) 算術ユニットと、
b) 乗算累算器と、
c) 論理オペレータと、
d) バレルシフタとのうちの少なくとも1つを備える、請求項25に記載のプロセッサ。

【請求項27】 前記モード手段は前記少なくとも1つのオペランドにあるビットフィールドによって特定される、請求項1に記載のプログラマブルプロセッサ。

【請求項28】 前記モード手段は、前記少なくとも1

つのオペランドが選択される方法によって特定される、請求項1に記載のプログラマブルプロセッサ。

【請求項29】 前記モード手段は、前記少なくとも1つのオペランドのアドレスをさらに特定するメモリ位置にあるビットフィールドにおいて特定される、請求項28に記載のプログラマブルプロセッサ。

【請求項30】 前記モード手段に応答し、第2のオペランドがスカラモードにある状態で、第1のオペランドの各それぞれの要素をベクトルモードで処理するための第3のベクトル手段をさらに含む、請求項2に記載のプログラマブルプロセッサ。

【発明の詳細な説明】

【0001】

【発明の分野】この発明は信号プロセッサに関し、より特定的には空間並列処理能力を備えたデジタル信号プロセッサに関する。

【0002】

【発明の背景】近年、コンピュータ技術において、単一命令多重データ (SIMD) などの並列処理方式を備えるコンピュータは徐々に認識される割合を獲得してきている。SIMDコンピュータは概念的には図1(a)で示すことができるものであり、ここでは複数の処理要素 (PE) が1つのメインシーケンサによって監視されている。すべてのPEはメインシーケンサから通信される同じ命令を受取るが、別々のデータストリームからの異なったデータの組に対して動作する。図1(b)に示すように、各PEはそれ自身の局所メモリを備える中央処理装置 (CPU) として機能する。したがって、SIMDコンピュータは各PEのCPUとともに複数の同期された算術論理ユニットを用いることによって、空間的並列性を達成することができる。一旦データが各PE内に存在するようになれば、個々のPEがそのデータを扱うことは比較的容易なことであるとはいえ、相互接続 (図示せず) を介してすべてのPE間で分配および通信を行なうことは、極めて複雑な仕事である。よって、SIMDマシンは、通常専用とすることを念頭において設計されており、プログラミングやベクトル化の難しさのために、これらのマシンは汎用の用途には望ましくないものとなっている。

【0003】一方で、SPARC (登録商標)、PowerPC (登録商標)、および68000ベースのマシンなど現在の汎用計算機は、典型的には高性能グラフィック処理に際してはそれらの持つ32ビットメモリ空間をフルに利用してはいない。たとえば、これらのマシンのバスが32ビットの幅であるのに、映像および画像情報についてはデータは未だに16ビット幅または8ビットピクセルで処理されるように制限されている。しかしながら、これらの汎用マシンは高級言語のソフトウェア環境におけるプログラミングの便利さのため魅力的である。したがって、デジタル信号処理に応用されるような

SIMDのスピードの利点と、汎用CPUにおけるプログラミングの便利さとの間で、バランスをとることが望ましい。そうすれば、低性能なSIMDマシンの実現例であっても、汎用マシンに組込まれたならば、あたかも複数のスカラCPUが並列に働いているかのように総合的なスループットが激しく向上するであろう。しかしながら、汎用マシンにSIMDが組込まれた場合、高められたスループットは、伝統的なSIMDマシンに見られる複数ユニットのスカラCPUと典型的にはかかわりのある、シリコンの使用と引替えにもたらされるものではない。

【0004】したがって、コード強調応用およびスピード強調計算のためのSIMD能力を備える汎用プロセッサを有することが望ましいだろう。

【0005】本発明の目的は、SIMD方式を汎用CPUアーキテクチャに組入れてスループットを高めることである。

【0006】実質的にシリコンの使用を招くことなくスループットを高めることも本発明の目的である。

【0007】この発明のさらなる目的は、同じ命令実行速度で各命令において処理されるデータ要素の数に比例してスループットを増大させることである。

【0008】

【発明の概要】SIMD方式を汎用プログラマブルプロセッサに組入れるための空間ベクトルデータ経路が開示される。プログラマブルプロセッサは、命令手段に結合され、オペランドがベクトルおよびスカラモードの1つにおいて処理されるかどうかを各命令について特定するためのモード手段と、モード手段に結合され、オペランドを受取り、モード手段によって特定された命令にตอบสนองして、オペランドをベクトルおよびスカラモードのうち1つにおいて処理するための処理ユニットとを備え、ベクトルモードは処理ユニットに、オペランド内に複数個の要素があることを示し、スカラモードは処理ユニットに、オペランド内に1つの要素があることを示す。

【0009】本発明はまた、汎用コンピュータを用いて複数のデータ経路を介しデジタル信号処理を行なう方法をも開示するものであって、汎用コンピュータは、各オペランドが少なくとも1つの要素を有する状態で複数個のオペランドをストアするためのデータメモリと、複数個のサブ処理ユニットを有する処理ユニットとを含む。この方法は以下のステップを含む。a) 処理ユニットによって実行されるべき命令の予め定められたシーケンスの中から命令を提供する。b) 命令はオペランドに対する処理ユニットによる処理についてスカラモードおよびベクトルモードのうち1つを特定する。スカラモードはオペランド内に1つの要素があることを処理ユニットに示し、ベクトルモードは複数個のサブ要素がオペランド内にあることを前記処理ユニットに示す。c) スカラモードの場合、処理ユニットにおける各サブ処理ユニットは

命令にตอบสนองして処理すべきオペランドのそれぞれの部分を受取り、部分的中間結果を発生する。d) 各サブ処理ユニットは複数のサブ処理ユニット間にその中間結果を送り、その部分的結果を他のサブ処理ユニットと合せて、オペランドのための最終的な結果を発生する。e) 最終的な結果に対応するように第1の条件コードを発生する。f) ベクトルモードの場合、処理ユニットにおける各サブ処理ユニットは命令にตอบสนองしてオペランド内の複数個のサブ要素からそれぞれのサブ要素を受取りかつそれを処理して、部分的中間結果を発生し、各中間結果は不能化され、各部分的結果はその対応する要素のための最終的な結果を表わす。g) 複数個の第2の条件コードを発生する。ここで第2の条件コードの各々は独立した結果に対応する。

【0010】

【発明の詳しい説明】

一般的な実現例の考察

SIMD方式を汎用マシンに組込む場合、望ましくは考慮されるべきである問題がいくつかある。

【0011】1) スカラまたはベクトルの動作の選択は、好ましくは、ある期間ベクトルモードに切り換わるのではなく、命令単位で行なわれるべきである。なぜなら、いくつかのアルゴリズムはベクトルサイズが大きいと容易にベクトル化されないからである。また、ベクトル演算が選択される場合、ベクトルの次元を特定しなければならない。

【0012】現在、本発明に従い、スカラ/ベクトルについての情報はSIMD能力を有する各命令内のデータタイプ修飾子フィールドによって特定される。たとえば、命令はワードまたはハーフワード演算を特定することのできる1ビット「経路」修飾子フィールドを特徴としていてもよい。さらに、より大きいベクトル次元、たとえば4、8などを選択するために、このフィールドは好ましくはストリーマコンテキストレジスタ内のデータタイプ変換フィールドと組合せられるべきである。ストリーマの完全な説明は、「RISCデジタル信号プロセッサのためのストリーマ (STREAMER FOR DIGITAL SIGNAL PROCESSOR)」と題され、その開示がここに引用によって援用される、1992年7月23日に提出された関連の米国特許出願連続番号第917,872号に開示されている。

【0013】2) マシンは、ベクトル結果に基づく条件付実行に備えるものでなければならない。SIMD演算の結果を、それがちょうど多重スカラ演算を用いて行なわれたかのようにテストできることが重要である。この理由により、ステータスレジスタ内の条件コードフラグは、データ経路の1セグメントごとに1組が存在するように二重にされることが好ましい。たとえば、ベクトル次元が4であれば4組の条件コードが必要であろう。

【0014】また、条件付命令は、条件コードのどの組

を使用するかを特定することを必要とする。たとえば「1つでもけた上げフラグがセットされていれば」または「すべてのけた上げフラグがセットされていれば」などの条件の組合せをテストすることができれば有用である。

【0015】3) SIMD方式は可能な限り多くの演算に応用可能であるべきである。これから述べる本発明の好ましい実施例は、16ビット乗算器および32ビット入力データなどの現在の実現例におけるマシンを示しているが、本発明に従い他の変形が容易に構成され得るということは当業者には認識されるであろう。

【0016】次の演算は、空間ベクトル(SV)技術の性能を高めることができる、可能な演算(図28~36で列挙)の例である。

【0017】ABS, NEG, NOT, PAR, REV, ADD, SUB, SUBR, ASC, MIN, MAX, Tcond
SBIT, CBIT, IBIT, TBZ, TBNZ
ACC, ACCN, MUL, MAC, MACN, UML, UMAC
AND, ANDN, OR, XOR, XORC
SHR, SHL, SHRA, SHRC, ROR, ROL
Bcond
LOAD, STORE, MOVE, Mcond
ここでcondは、CC, CS, VC, VS, ZCおよびZSであってもよい。

【0018】4) メモリデータ帯域幅はSIMDデータ経路の性能に適合可能であるべきである。

【0019】メモリおよびバス帯域幅をハードウェアの複雑さを増大させることなく空間ベクトルデータ経路のデータ要求に適合させることが望ましい。現在実現されているマシンにおけるデュアルアクセスの32ビットメモリを備える2つの32ビットバスは、算術論理ユニット(ALU)およびデュアル16×16乗算/累算ユニット(MAC)によく適合している。これらはまた、4つの8×8MACにもよく適合するだろう。

【0020】5) 実現されるいかなる付加および変形も、付加的なハードウェアの複雑さは最小限で性能を最大限にすることによって、コスト効率の良さをもたらしべきである。

【0021】加算器/減算器は、けた上げ伝播を止め、条件コード論理を二重にすることによって、空間ベクトルモードにおいて動作させることができる。

【0022】シフトは、ラップアラウンド論理をも再構成し、条件コード論理を二重にすることによって空間ベクトルモードにおいて動作させることができる。

【0023】ビット論理ユニットは、条件コード論理を二重にするだけで空間ベクトルモードにおいて動作させることができる。

【0024】空間ベクトル条件付移動動作は、条件コー

ドフラグのベクトルを用いてマルチプレクサを制御し、ベクトルの各要素が独立的に移動させられるようにすることによって達成され得る。

【0025】空間ベクトルの乗算は、乗算器アレイを二重にし、部分積を組合せることを必要とする。たとえば適切な組合せ論理を備える4つの16×8乗算器は、4つの16×8または2つの16×16のベクトル演算、もしくは1つの32×16スカラ演算を行なうのに用いることができる。空間ベクトル乗算-累算演算はまた、けた上げ伝播を止め、条件コード論理を二重にすることができる累算加算器を、ベクトル化された累算器レジスタと同様に必要とする。

【0026】6) 汎用コンピュータにおける空間ベクトルの実現に起因するプログラミングの複雑さは、最小限にされるべきである。空間ベクトル結果をスカラ結果に組合せるために命令を考え出すことができる。

【0027】ACC Az, Ax, Ay 累算器を加算する。SA Ay, Mz スケーリングされた累算器対をメモリにストアする。

【0028】MAR Rz, Ax スケーリングされた累算器対をレジスタに移動させる。7) ベクトルが物理的メモリ境界と交差するとき、ベクトルへのアクセスがそれでも可能であるべきである。たたみ込みなどのいくつかのアルゴリズムは、データアレイを介しての増分を必要とする。アレイが長さNのベクトルとして扱われる場合、ベクトルが部分的に1つの物理的メモリ位置の中に存在し、部分的に隣接する物理的メモリ位置の中に存在するということがあり得る。そのような空間ベクトル演算に対する性能を維持するには、物理的境界と交差するデータアクセスに対処するようにメモリを設計するか、または前述の米国特許出願「RISCデジタル信号プロセッサのためのストリーマ」に記載されたようなストリーマを用いることが好ましい。

【0029】全体システム

図2は、本発明の空間ベクトルデータ経路を組入れてもよいプログラマブルプロセッサを一般化して表わしたものである。本発明に取入れられたコンセプトの1つは、スカラオペランドまたはアレイの要素に一度に1つずつ対処するように設計されたコンピュータを変形し、同時に1つより多くのオペランドを処理できるようにすることによって、その性能を高めることができるということである。

【0030】図2に示されているのは、プログラムおよびデータオペランドをストアするためのプログラムおよびデータ記憶ユニット100を有するプログラマブルプロセッサ、または広い意味でいう「コンピュータ」である。命令収集ユニット130が記憶ユニット100から命令をフェッチし、これは命令フェッチ/デコード/シーケンスユニット140によりデコードかつ解釈され、処理ユニット110によって実行される。このようにし

て処理ユニット110は記憶ユニット100から供給されるオペランドで命令を実行する。

【0031】性能を高めるため、オペランドがスカラであるかベクトルであるかを特定するためのビットが各命令の中にある。また、それらがベクトルである場合、各オペランド内にいくつの要素があるかが特定される。この情報は典型的なデコードされた命令とともに処理ユニット110に送られるので、処理ユニット110はオペランドをスカラとして処理するべきかベクトルとして処理するべきかを「知る」。

【0032】処理ユニット110はALUでもシフトでもMACでもよい。記憶ユニット100は一般に何らかの種類のメモリであってよく、レジスタファイルでも、半導体メモリでも、磁気メモリでも、またはいくつかの種類のメモリのいずれのものでもよい。処理ユニット110は加算、減算、論理AND、論理OR、バレルシフトでのようなシフト、乗算、累算、およびデジタル信号プロセッサにおいて典型的に見られる乗算および累算のような、典型的な演算を行なってもよい。処理ユニット110はオペランドを、命令において用いられる1つのオペランド、命令において用いられる2つのオペランド、またはそれ以上多くのものなどのうちいずれかとしてとる。処理ユニット110は次にこれらのオペランドで演算を行なっており、それらの結果を得る。スカラまたはベクトルオペランドで開始することにより、オペランドは演算を最後まで行なわれ、それぞれスカラまたはベクトル結果をもたらす。

【0033】次のステップは、処理ユニット110がどのように形成されてもよく、どのように機能するかをより特定の認識するためのものである。データおよびプログラムは記憶ユニット100内で組合せられているように示されているが、それらは同じ物理的メモリ内で組合せることもできるし、別個になった物理的メモリ内で実現することもできるということは明らかであろう。各オペランドは典型的な32ビットの長さを有するものとして説明されているが、一般に、オペランドはいくつかの長さのいずれとすることもできるだろう。16ビットマシン、8ビットマシン、または64ビットマシン等々とすることができる。一般的なアプローチは、Nビットオペランドが、ともにとられて加算されるとNビットになる複数オペランドとして考えられ得るということであると、当業者は認識するだろう。したがって、32ビットワードはたとえば2つの16ビットハーフワード、もしくは4つの8ビットクォーターワードまたはバイトであり得るだろう。発明者らによる現在の実現例では、1つのオペランド中の要素は各々同じ幅のものとしている。しかしながら、32ビットオペランドを、一方の要素を24ビットとし、他方の要素を8ビットとすることもできる。オペランド中で複数のデータ経路および複数の要素を用いることから導き出される利点とは、すべて

の要素が独立的かつ同時に処理されており、処理のスループットの増加がなし遂げられるということである。

【0034】命令はどのようなサイズであってもよい。現在は32ビット命令が用いられている。しかしながら当業者は、8ビット、16ビット、32ビット、および64ビットにおいて特に有用性を見出すかもしれない。より重要なことは、命令については固定長でさえなくともよいということである。同じコンセプトが、32ビット命令に拡張可能な16ビット命令を備えるものなどの、または命令がいくつかの数の8ビットバイトで形成されており、その数はそれがどの特定の命令であるかによって決まる、可変長命令マシンにおいて用いられた場合でも、働くだろう。当業者のために、図28~36に例示的な命令セットのまとめを示し、本発明に従って実現されてもよい命令を示す。

【0035】処理ユニット110は典型的にはALU121および/またはMAC122を含んでもよい。またこれは、シフト123または論理ユニット124を実現するだけのものであってもよい。

【0036】加算器

図3は処理ユニット(図2の110)のための、ALUにおいて実現されてもよい加算器を模式的に表わしたものである。図3(a)は従来の32ビット加算器を示す。図3(b)はハーフワード対モードのために接続された2つの16ビット加算器を表わしたものである。図3(c)はワードモードのために接続された2つの16ビット加算器を表わしたものである。

【0037】図3(a)から(c)は、図3(a)における32ビットの従来のマシンにおける典型的なハードウェアが、本発明に従うハーフワード対モードまたはワードモードの所望される目的を達成するためにどのように変形されてもよいかということを示す役割を果たす。ベクトルはここでは2つの要素を持つものとして示される。より特定のには、32ビットの従来のオペランドがどのようにして各々16ビットの2つの要素に分割され得るかということが示される。同じ原理を、等しい長さまたは等しくない長さのものがあるいくつかの要素に分割するのに適用することができるだろう。

【0038】図3(a)を参照して、従来の加算器200はXオペランドのための入力XとYオペランドのための入力Yとを有する。またこれは、加算器と関連して典型的に見出されるキャリーイン201および条件コード205のための入力をも有する。条件コード205は、オーバフローを表わすのがV、キャリーアウトがC、ゼロ結果、すなわち加算器から出される結果がゼロである場合がZであってよい。さらにこれは加算器から出される結果オペランドを有しており、これはSである。X、Y、およびSはすべて32ビットワードで表わされる。制御入力s/u202は符号付または符号なしオペランドを表わし、ここで最上位ビットはその数が正

または負である場所を示し、もしくは符号なしオペランドではその最上位ビットがオペランドの大きさに関与する。図3(b)は、典型的な32ビット加算器に類似してはいるが、そうではなく単なる16ビット加算器である2つの加算器が、どのようにともに組合せられてハーフワード対、すなわち1つのオペランドにつき2つのハーフワード要素があるものに対してベクトル演算を行なうことができるかということを示す。Yオペランドはここでは2つのハーフワードオペランド、すなわち下半分のY0からY15、および上半分のV0からV15として分割されている。同様に、Xオペランドは2つのハーフワードオペランド、すなわち下半分のX0からX15、および上半分のU0からU15として分割されている。結果Sは、加算器210からくるS0からS15、および加算器220からくる上半分のW0からW15として認識される。本質的には、32ビット加算器200を中央で分割して2つの16ビット加算器210および220を形成してもよい。しかしながら、上位ビットにはオペランドの符号ビットの性質を決定するための論理が必要であろう。したがって32ビット加算器200を分割する際には、32ビット加算器から分割されて加算器210を形成する下方の16ビットの符号制御のために付加的な論理が必要となるであろう。この場合これら2つの加算器210および220は、加算器210のための入力オペランドが32ビットオペランドの下半分からきており、16ビット加算器220のための入力オペランドが32ビットオペランドの上半分からきているということを除けば、同一なものとなるであろう。

【0039】オペランド要素XおよびUが別個にそれぞれYおよびVと加算されて合された場合、それらはそれぞれ結果SおよびWをもたらす。またそれらは加算器の各々のために独立な条件コードを生成する。加算器210は条件コード215を生成し、加算器220は条件コード225を生成する。これらの条件コードは、それらが関連している特定のハーフワード加算器に適用される。したがって、これで独立なハーフワード対演算を行なうために従来の32ビット加算器がわずかに変形される様が見てとれる。

【0040】図3(c)を参照して、図3(b)における同じ加算器ユニットが、図3(a)の加算器200において行なわれた、もとのワード演算を行なうべく再接続されてもよい。これは、オペランドが32ビットスカラを表わす場合である。スカラはY0からY31およびX0からX31である。これらのオペランドの下半分は加算器230によって処理され、上半分は加算器240によって処理される。これを可能にするメカニズムは、加算器230のキャリーアウトを加算器240のキャリーイン236に接続することによるものである。図2(c)に示されるように、組合せられた2つの16ビット加算器は図3(a)の1つの32ビット加算器と同

じ機能を果たす。したがって、図3(b)および3

(c)に示した実現例では、加算器210は本質的に加算器230と同じものであってもよく、一方で加算器220は加算器240と同じものであってもよい。この説明ではこれら2つの加算器がどのようにハーフワード対モードまたはワードモードのいずれかで機能できるかが示されているが、当業者は、拡張によってベクトルの独立した要素を同時に扱うために従来の加算器をいくつかの加算器に変形すること、およびこれを再結合してスカラ演算をスカラオペランドで行なうことをしてもよい。

【0041】図3の加算器について、1つ注目すべきことがある。図3(c)では2組の条件コード235および245が示されている。一方、もとの従来の加算器では1組の条件コード205しかない。図3(c)の条件コードは、本当は条件コードZを除いては245の条件コードである。235における条件コード、すなわちオーバーフローVおよびキャリーCは、条件コード205における条件コードおよび条件コードZが、効果的に235のZ条件コードとAND処理される245のZ条件コードである限り、無視される。ここでは205の条件コードVは245のVに対応する。205のCは245のCに対応し、205のZはコード235のZとAND処理されたコード245のZに対応する。当業者は、適合すると思われるなどの特定のやり方でもこれらを組合せることができるだろう。

【0042】論理ユニット

図4は本発明に従い実現されてもよい論理ユニットの模式図である。図4(a)はビット単位の論理演算、ビット単位の補数または現在のプロセッサにおいて典型的に見られるいくつかの組合せを行なう典型的な32ビット論理ユニットを示すものであって、これらの演算について重要なかもしれないのは、それらが条件コードにおける異なったビットのために独立に働くということである。オーバーフロービットは通常、305における条件コードでは全く重要性を持たない。キャリーアウトは論理演算においてまったく重要ではないが、ゼロには、結果がゼロであるということを示すことにおいてまだ重要性がある。ハーフワード対演算のために、もとの32ビット加算器は「動作的」には2つの16ビット論理ユニットに分割されるだろう。入力オペランドにおける上方の16ビット320および下方の16ビット310は、加算器のときと同じ態様で2つのハーフワードに分割されるだろう。論理演算を処理するにあたっては、ビットは一般に独立的に処理されるので、2つの論理ユニット310および320の間には動作的な接続は全くない。

【0043】図4(c)はスカラ処理のための典型的な論理ユニットを形成するようにもう一度再結合された論理ユニットを示す。条件コードエリア以外ではユニット間には接続が必要ではないということに注意されたい。従来の論理ユニットのゼロ条件コード305はここでは

ユニット345のゼロ条件コードをユニット335のゼロ条件コードとAND処理することによって表わされてもよい。したがって当業者には、デュアルモード論理ユニットが前述のようにデュアルモード加算器のコンセプトおよび実現例を拡張することによって構成され得るということが明らかなはずである。

【0044】シフタ

図5から8は、本発明に従い実現されてもよいバレルシフタを模式的に表わしたものである。いくつかのプロセッサは図5(b)に示されるようにバレルシフタを有するが、他のものは図5(a)、図6、および図7に示される1ビットシフタを有する。バレルシフタは典型的にはプロセッサユニット内に必要なものではないが、高性能マシンについては、プロセッサユニットは図5(b)に表わされるようなシフタユニットを実現してもよい。以下の説明では、処理を高速化する、または必要なハードウェアの量を最小限にするために、当業者によってシフタがどのように構成され実現されてもよいを示す。図5(a)は、左シフトまたは右シフトのどちらかである1ビットシフトが典型的なプロセッサにおいてどのように実現されてもよいを示す。シフタ415は、32ビット入力オペランドXが、左または右へ1ビットシフトされる、または方向入力DIR401の制御下ではシフトされないようにして、Z出力を生成できる。シフトが起こった場合、それが左へのシフトなら、選択ボックス416によって最下位ビットの位置にビットが入れられなければならない。

【0045】シフタが右へシフトされる場合、選択ボックス400からのビットが最上位ビットの位置に入れられる。選択ボックス400および416はシフタ415に入れるために選択され得るいくつかの入力を有する。双方のボックスにはSELとラベル付けされる選択入力もあり、これは命令からくるものであって、従来のマシンには典型的なものである。SELはこれらの入力ビットのうちどちらがシフタに入れられるために選択されるであろうかを決定する。一般に、これらの選択ボックスがあるため、シフトは、シフタの外へシフトされるビットがシフタのもう一方の端で中にシフトされる回転でもあり得るし、他のビットが右へシフトされる際に符号ビットまたは最上位ビットがドラッグされる算術的右シフトでもあり得るし、他のビットが左へシフトされる際に0が入れられる算術的左シフトでもあり得る。論理シフトについては、「0」がビットとして入れられる。また、「1」は論理シフトに入れられる新しいビットとして、入れられる。

【0046】当業者は、加算器および論理ユニットのための条件コードの説明を参照することによって、容易に条件コードをシフタに割当て、算術的左シフト演算のためのオーバフロー、シフト演算の最後のビットを保持するためのキャリー、およびシフトの結果が0値であった

ときにそれを記録するゼロフラグを表わすことができるだろう。

【0047】図5(a)のシフタを組合せて用いることで、図5(b)のシフタを32ビット左/右バレルシフタとして形成してもよい。これは図5(a)におけるシフタを32個組合せ、それらを次々にカスケード接続し、第1のものの出力が第2のものの入力に入る、というふう到最后まで続いていくようにすることによって行なわれてもよい。シフトされるべきビットの数は個々のシフタへの方向入力DIRの1および0のパターンによって決定される。図5(a)ではシフタのための方向は3値であるということに注意されたい。すなわち左、右、またはまったくシフトがなし遂げられない真っ直ぐ前方、である。そこで図5(b)では、個々の32ビットの1ビットシフタへの方向入力は、左でも右でもシフトなしでもあり得る。32ビットが左へシフトすべきである場合、すべての方向入力が左を示すだろう。

【0048】左へシフトすべきなのが1ビットだけの場合、第1のボックスが左への1ビットシフトを示し、他の31個はすべてシフトなしを示す。Nビットが左へシフトすべきである場合、始めのN個のボックスが左への1ビットの方向入力を有し、残りのボックスがシフトなしを示すだろう。同じことが右へのシフトにも適用できるだろう。この場合には方向は右へのシフトまたはシフトなしのいずれかを示し、同じように右シフトにおいて0ビットから32ビットまでのシフトが可能であろう。

【0049】この図5(a)における典型的な1ビットシフタは、ここで図6を参照して2つの16ビットシフタに分割することができる。ここではハーフワード対モードのために接続された2つの16ビットL/R1ビットシフタが示される。図5(a)におけるシフタ415は、動作的には2つの16ビットの1ビットシフタ450および435に分割できる。これらの16ビットシフタの各々は、この場合特に416および400を参照する図5(a)に示す入力選択論理を有しており、これはボックス450がボックス460および445を有し、ボックス435がボックス440および430を有するように二重にされる。入力論理は同じであるが、選択ボックスへの入力は異なったように結線される。したがって、ハーフワード対モードのために接続される図6のシフタとワードモードのために接続される図7のシフタとの違いは、入力選択ボックスの結線のされ方にある。下方のシフタ450のための図6の入力オペランド要素はX0からX15であり、シフタ435のための入力オペランド要素はY0からY15である。このようにしてXおよびYは2つのハーフワードを示す。

【0050】結果Z出力オペランドは2つのハーフワードとして示される。下方の16ビットはZ0からZ15であり、上方の16ビットはW0からW15である。入力セレクトは、回転においてシフタから出力されるビッ

トがシフタの他方の端にフィードバックされるように結線される。シフタ 435 が左シフトを行なうと、回転されるビットは Y15 となり、右シフトを行なうと回転されるビットは Y0 となる。同様にシフタ 450 について、それが左回転であれば、入力ビットは X15 であり、右回転であれば入力ビットは X0 である。同様に、選択は算術的シフトおよび論理的シフトについても図 5 (a) のように働く。

【0051】図 7 は、これらの同じ 2 つのシフタの動作がどのようにワードモードのために接続され得るかを示す。ここではシフトパターンは図 6 での 2 つのハーフワードとは違ってオペランドにおける 32 ビット全体に対して働く。左への回転については、下方のシフタ 486 から外へ回転させられるビット (MSB ビット X15) は上方のシフタ 475 の中に回転させられる一方で、LSB ビットはシフタ 475 に入力される。これは上方の 1 ビットシフタと下方の 1 ビットシフタとの間で連続的なシフトを形成する。2 つのシフタをめぐる回転については、X31 が X0 にシフトされるだろう。図 7 に示したようにセレクト 480 のすべての入力が X15 に接続されており、セレクト 485 のすべての入力が X16 に接続されていれば、図 7 の組合せられたシフタは図 5

(a) におけるシフタとして効果的に動作する。入力セレクト 470 は入力セレクト 400 と同じパターンを有するだろう。入力セレクト 488 はセレクト 416 と同じ入力パターンを有するだろう。したがって、図 7 の組合せられたシフタは図 5 (a) におけるシフタと同じスカラオペランドのためのシフト動作を行なうだろう。

【0052】図 6 および 7 における 1 ビットシフタはさらに、1 ビットシフタを 32 個カスケード接続することによって、図 5 (b) と類似の態様で、図 8 に示した 32 ビットバレルシフタに拡張することができる。1 ビットシフトが所望されるならば、方向制御信号が第 1 のシフタに対して用いられ、1 ビットシフトを示す。他のカスケード接続された 1 ビットシフタに対しては、示されるシフトはない。N ビットシフトについては、最初の N 個の 1 ビットシフタにおける方向入力が、1 ビットだけシフトすることを示し、残りの 1 ビットシフタはシフトせずデータを通過させる。

【0053】同様にこの図 8 のバレルシフタはワードまたはハーフワード対モード演算のいずれをも行なうことができる。なぜなら、個々のビットシフタはワードまたはハーフワード対演算のどちらでも行なうことができるからである。この図 5 から 8 の実施例はバレルシフタを実現する 1 つの方法を代表するものであるが、バレルシフタを真中で分割して入力選択論理を提供する同じコンセプトが、バレルシフタの多くの他の実現例にも応用できる。当業者は、特定のハードウェアまたはスルーブットの要求に応じて適切な実現例を見出すことができるはずである。

【0054】乗算累算器

図 9 および 10 は、本発明に従い実現されてもよい乗算および累算 (MAC) ユニットの模式図である。

【0055】典型的な 32 ビットプロセッサは通常、高価な 32×32 乗算器アレイの実現を必要とはしないだろう。乗算はおそらく他の方法で確立されるだろう。しかしながら典型的な 16 ビット信号プロセッサでは、 16×16 乗算器アレイが極めて普通に見られる。高速な乗算を必要とするタイプの計算には、典型的に 16 ビットデータが用いられるので、 16×16 乗算器アレイの方が普及したものとなっており、これはいくつかの 32 ビットプロセッサにおいてさえ当てはまることである。したがって、32 ビットオペランドを 2 つの 16 ビットハーフワード対として扱うことにより、1 つのベクトル化されたオペランド中の 32 ビットワードオペランド、ハーフワードオペランド、またはハーフワード要素の空間ベクトルの概念を利用すべく 2 つの 16×16 乗算器アレイを実現することができる。

【0056】次の例は、どのようにして 16×16 乗算器アレイを二重にして 2 つのハーフワード対乗算器として用いることができるかを示すものであって、これらの乗算器はともに接続されて 32×16 スケラ乗算をもたらしてもよい。この 32×16 スケラ乗算には、これらの乗数の 2 つを一緒に用いて 32×32 ビット乗算をなすことができるという有用性がある。または、 32×16 乗算をそれ自体で用いることもでき、この場合 32 ビットの精度のオペランドがただ 16 ビットだけの精度のオペランドによって乗算されてもよい。

【0057】MAC ユニットはすべてのプロセッサで典型的に見られるわけではない。しかし信号処理の用途のための高性能プロセッサでは、これは典型的に実現されている。図 9 は、MAC ユニットの従来の実現例を示す。MAC は様々なサイズのうちのどのサイズでもあり得る。これは 32 ビットの積を形成する乗算器における 16 ビット \times 16 ビットのユニットである。この 32 ビットの積は累算加算器内で第 3 のオペランドと加算されてもよく、これは「ガードビット」と呼ばれる余剰の上位ビットがあるためその積よりも長いかもしれない。

【0058】図 9 に示されるように、入力オペランドは 16 ビットであって、X0 から X15 および Y0 から Y15 で表わされる。これらは 32 ビットの積 Z を発生し、これはフィードバックオペランド F に加えられてもよい。この場合、F は 40 ビットのフィードバックワードまたはオペランドを表わす F0 から F39 として示される。これが 40 ビットなのは、積を保持するのに 32 ビット、加えてガードビットのために付加的な 8 ビットが必要とされるであろうからである。ガードビットはオーバーフローを扱うために含まれている。なぜなら、いくつかの積が加算されると、オーバーフローが起こる可能性があり、ガードビットはオーバーフローを累算してそれら

を保護するからである。典型的にはガードビットの数は4または8であろう。この例では8ビットが示されているが、いくつかのサイズが可能であろう。累算器の結果は40ビットの結果A0からA39として示される。

【0059】乗算アレイは乗算器なしで用いることもできるし、乗算器とともに用いることもできるということに注意すべきである。符号付または符号なしを意味する別の入力S/Uが、入力オペランドが符号付数として扱われるべきか符号なし数として扱われるべきかを示すということが注意されるべきである。当業者は、乗算器の上方のビットが、入力オペランドが符号付であるか符号なしであるかによって異なったように扱われるということ認識するであろう。

【0060】図10は、ハーフワード対オペランドを扱うためにどのように典型的な16×16アレイが形成されるかを示す。この場合、32ビット入力オペランドXが2つのハーフワードに分割される。乗算器520のための下方のハーフワードはX0からX15であり、乗算器515のための上方のハーフワードはX16からX31である。Y入力オペランドも2つのハーフワードオペランドに分割される。乗算器520のための下方のハーフワードはY0からY15であり、乗算器515のための上半分はY16からY31である。図10はこれのようにしてXオペランドのハーフワードオペランドをそれぞれYオペランドのハーフワードオペランドと乗算するための接続を表わす。Xの最下位ハーフワードは乗算器520においてYの最下位ハーフワードと乗算されるということに注意されたい。また乗算器515において独立かつ同時に、Xの上方のハーフワードがYの上方のハーフワードと乗算される。これらの2つの乗算は、2つの積を生じる。乗算器520からの32ビットの積はZ0からZ31で表わされ、同様に乗算器515の32ビットの結果はW0からW31によって表わされる。2つの積はその精度を保つために各々16ビットよりも大きい。この時点で、ハーフワードの積は独立したオペランドの表現として保存される。

【0061】乗算器520より出された下方のハーフワードからの積は、累算器530に送られF0からF39で表わされるフィードバックレジスタで加算される。これにより、A0からA39で表わされる累積された積Aが形成される。同様に上方のハーフワードにおいて、積はW0からW31によって表わされており、かつ累算器525の中でG0からG39によって表わされるフィードバックレジスタに加算されて40ビットの結果Bを形成し、この結果はB0からB39で表わされる。これらの累算器の結果は一般に、乗算の精度を保つためにより大きい数またはビットで表わされるオペランドとして累算器の中でより大きい数として保存される。

【0062】フィードバックビットは通常、メモリ(図2の100)またはより大きい数のビットをストアする

ことのできる特殊なメモリのいずれからでももたらされるだろう。典型的なメモリ位置が扱えるのは32ビットであるが、典型的には累算器ファイルと呼ばれる特殊なメモリは、スカラ積のために40ビット、またはハーフワード対の積のために80ビットをストアすることができるだろう。この場合スカラオペランドを扱うことのできる2つの累算レジスタが、ハーフワード対オペランドのための記憶を形成するのに用いられてもよい。換言すれば、ハーフワード対演算の2つの40ビットの結果をストアするのに2つの40ビット累算器を用いることができるだろう。

【0063】MAC相互接続

図11および図12は、スカラオペランドのための16×32ビット乗算を形成するために、図10のアレイの2つの16ビット乗算器がどのように相互接続され得るかを示す。この例では、乗算器アレイは加算器列として実現される。最下位乗算器アレイ610のキャリーアウト605は、上位乗算器アレイ600の加算器にキャリー入力として与えられる。さらに、上位乗算器アレイ600の最下位端に形成される合計ビット606は、下位乗算器アレイ610の加算器の最上位端に与えられる。

【0064】他の接続は、累算器615および605に生ずる。積の下位部分を表わす累算器615は32ビットに制限され、上位8ガードビットは使用されない。32ビットのキャリーアウトは上位40ビット累算器605のキャリー入力に与えられ、その結果はB39を通るB0としてここでは示される72ビットオペランドである。典型的にはこのオペランドは2つのオペランドとしてストアされ、下位32ビットは1つの累算器615にストアされ、上位40ビットは第2の累算器605にストアされる。さらにこの演算では、符号付ビットおよび符号なしビットのために、入力オペランドXの下位半分は乗算器610において符号なし数として扱われ、入力オペランドXの上位16ビットは上位乗算器アレイ600において符号付または符号なしオペランドとして扱われる。

【0065】さらに、下位累算器615においては積は符号なしオペランドとして扱われ、一方上位累算器605ではオペランドは符号付数として扱われる。40ビット累算器は図11および図12のすべての例においては符号付数として扱われることを付け加えるべきである。これは、符号なし数でさえも符号付数の正の部分と考えられ得るようなビットを、累算器の拡張であるガードビットが可能にするからである。ゆえに、拡張累算器における符号付数は、符号付オペランドと符号なしオペランドの両方を含む。

【0066】図12は、乗算器アレイ600および610を構成する加算器間でキャリーおよび合計ビットがどのように相互作用するかをより詳細に示す。たとえば、

加算器 625 および 635 は乗算器 アレイ 610 の一部として示され、加算器 620 および 630 は乗算器 アレイ 600 の一部として示される。乗算器 アレイ 610 および 600 は典型的には加算器の何らかの構成でもって実現されることが注目されるべきである。特定の實現例において、加算器の相互接続は様々な方法でなされるであろう。図 10 は加算器の単純なカスケードを示すが、この同じ技術を、加算器がたとえばブース乗算器またはウォレス・ツリー乗算器におけるように接続されるであろうような他の方法に用いてもよい。図 12 に示されるように、下位乗算器 アレイ 610 の加算器 625 は、上位乗算器 アレイ 610 の対応する加算器 620 のキャリー入力に与えられるキャリーアウト 621 を与える。下位乗算器 アレイ 610 は、X-入力の入力オペランドがあたかも符号なしであるかのように演算を行なう。入力オペランドの符号は特定されて、上位乗算器 600 アレイの上位加算器 620 および 630 の符号制御に用いられる。

【0067】さらに加算器は、それらが乗算器の最下位ビットから乗算器の最上位ビットにオフセットされるような方法で接続されるため、それは合計ビットを再び加算し戻す機会を与える。より特定のには、加算器 625 および 620 は、 Y_i とされる、乗算器のより下位のビットに対応する。加算器 635 および 630 は、 $Y(i+1)$ とされる、乗算器の次のより上位のビットに対応する。このオフセットは、加算器 635 の入力 B0 に与えられる加算器 625 の出力 S1、および加算器 635 の入力 B14 に与えられる加算器 625 の S15 として見られ得る。この 1 ビットのオフセットは加算器 620 からの入力 S0 を受取るよう加算器 635 の入力の B15 を解放して、最上位乗算器 アレイ 600 からの合計ビットは最下位乗算器 アレイ 610 へ入力ビットとして与えられる。

【0068】さらに、加算器 625 からの合計ビット S0 は、640 として示される次の部分積に直接進み、さらなる乗算器または加算器段を通る必要はない。したがって、連続する加算器段 625、635 などから S0 を出力することは、図 9 の出力ビット Z0 から Z15 を生じさせる。加算器 635 の S0 から S15 に対応する最終部分積からの出力ビットは、Z16 から Z31 の図 9 のアレイ 610 からの出力ビットを生じさせるであろう。

【0069】万一乗算器 Y が負である場合に補償を与えるために最終加算器段がどのように用いられ得るかについては、当業者ならば理解するであろう。

【0070】オペランドデータのタイプの分類

オペランドデータのタイプの分類に関してここで注目する。オペランドモードのタイプをスカラまたはベクトルとして特定するための 1 つのアプローチは命令にその情報を含むことであるが、代替的アプローチはその情報を

オペランドの付加的なビットにおいて付け加えることである。たとえば、オペランドが 32 ビットの場合、1 つの付加的なビットを用いて、オペランドをスカラまたはベクトルのいずれかとして識別してもよい。仮にベクトル要素の数がはっきりと示されるか、またはベクトル要素の数が 2 のような何らかの数であると仮定され得るような場合、付加的なビットがさらに用いられてもよい。オペランド処理ユニットは、オペランドに付加される情報に応答することによってオペランドをスカラとしてまたはベクトルとして処理するのに適合されるであろう。

【0071】オペランドがスカラであるかまたはベクトルであるかは、オペランドが選択される方法によってさらに特定されてもよい。たとえば、オペランドのアドレスをさらに特定するメモリ位置にあるビットフィールドに、情報が含まれてもよい。

【0072】2 つのオペランドが処理ユニットによって処理され、モード情報がその 2 つのオペランドにおいて異なる場合には、混合されたモード演算を処理するために当業者によって処理ユニットに規定が設定されてもよい。たとえば、ベクトルオペランドおよびスカラオペランドを伴う ADD 演算は、処理ユニットによって、スカラからベクトルを形成し、必要ならば切捨て、次いでベクトル演算を行なうことによって処理されてもよい。

【0073】空間ハードウェアに対する代替としてのタイムシェアリング

実現手段をタイムシェアリングすることは空間に分散する実現手段の代用にしばしばなり得ることは、当業者には理解されるであろう。たとえば、空間に分散されるベクトル処理ユニットにおいて多重加算器を効果的に実現するのに、1 つのベクトル加算器が何度も用いられてもよい。ハードウェアの多重化および非多重化を用いて、入力オペランドおよび結果を順序づけることも可能である。付加的なサポートハードウェアを有するベクトル加算器をさらに用いて、スカラオペランドを処理するのに分散ベクトル加算器が相互接続され得る方法と類似の態様でスカラオペランドをばらばらに処理することも可能である。サポートハードウェアは、ベクトル演算処理素子間を通る中間結果を処理するのに用いられる。

【0074】この発明の上記の説明に留意して、この発明の空間ベクトルデータ経路を組込む例示の RISC 型プロセッサがこれより説明される。以下のプロセッサシステムは、当業者がこの発明を組込むであろう方法の一例にすぎないことに注意されたい。他の例は、記載されるこの発明に基づく、それらの有利なアプリケーションを見出すであろう。

【0075】この発明を組込む例示的プロセッサ

この発明を組込む演算処理素子の機能図を示す図 13 を参照する。以下の説明は特定のビット幅を参照するが、それらは例示のためであり、この発明の教示に従って他の幅が容易に構成され得ることを、当業者は理解するで

あろう。

【0076】図13を参照すると、図示されるデータ処理ユニットを制御するために、2つのソースオペランドおよび1つの宛先オペランドを特定することのできる命令が用いられる。

【0077】オペランドは典型的にはレジスタにおよびデータメモリ(200)にストアされる。演算命令、論理命令、およびシフト命令がALU240およびMAC230においてレジスタ空間からのオペランドを用いて実行され、その結果はレジスタ空間に戻される。レジスタ空間はレジスタファイル220と幾つかの他の内部レジスタ(図示せず)とから構成される。レジスタ空間にストアされるオペランドは、32ビットワードまたはハーフワード対のいずれかである。オペランドは、ロードおよびストア命令によってレジスタ空間とメモリ200との間を、または既に記載したようにレジスタ空間と自動メモリアクセスユニットであるストリーマ210との間を往復する。

【0078】図14を参照すると、ALU240の機能ブロック図が示される。ALUは加算器410、420とバレルシフタ470とから構成される。一般に、ALU命令は、レジスタ空間から2つのオペランドをとり、レジスタ空間にその結果を書込む。ALU命令は、各クロックサイクルを実行することができ、ALUパイプにおいて僅か1つの命令クロックサイクルを必要とするだけである。

【0079】加算器410、420およびシフタ470は、ワードまたはハーフワード対オペランドを用いて演算を行なう。符号付オペランドは2の補数表記法で表わされる。現在、符号付、符号なし、小数、および整数オペランドが、ALU演算のための命令によって特定可能である。

【0080】加算器

加算器(410、420)はワードおよびハーフワード対で加算および論理演算を行なう。ハーフワード対演算の場合、加算器410、420は半分のものが2つあるものとして機能する。下半分420はハーフワード対の下位オペランド460を用いて演算を実行し、上半分410は同じ演算をハーフワード対の上位オペランド450を用いて実行する。ハーフワード対モードにある場合は、2つの加算器410、420は本質的に互いから独立している。32ビット論理ユニット440は、下の加算器420から上の加算器410へ情報を送り、2つの加算器がワードモードで動作しているときには情報を逆に送るために用いられる。

【0081】加算器演算は、2つのキャリー(CUおよびCL)、2つのオーバーフロー(VUおよびVL)、および2つのゼロ(ZUおよびZL)条件コードビットに影響する。CUはワード演算のための桁上げフラグであり、CUおよびCLはハーフワード対演算のための桁上

げフラグである。同様に、VUはワード演算におけるオーバーフローを示し、VUおよびVLはハーフワード対演算におけるオーバーフローを示す。

【0082】オーバーフローフラグに作用するオーバーフローは、加算器演算命令からおよびMACスカラ命令から結果として生じ得る。オーバーフローフラグは、実行された命令がたとえ結果を飽和したとしてもセットされる。一度セットされると、条件コードは、フラグをセットすることのできる別の命令があるまで変わらない。

【0083】飽和のない加算器演算命令がオーバーフローし、誤り例外が可能化されると、誤り例外要求が生ずる。飽和のあるオーバーフローおよび飽和のないオーバーフローを示すために、別個の信号がデバッグ論理に送られる。

【0084】バレルシフタ

図14を参照すると、1クロックサイクルの間に、バレルシフタは32ビット位置までのワードオペランドにあるすべてのビットを左または右のいずれにもシフトさせながら、ゼロ、オペランドの符号ビット、または加算器の上位桁上げフラグ(CU)を回転または挿入することができる。ハーフワード対演算の場合には、1クロックサイクルで、シフタは16ビット位置までの両方のハーフワードを左または右へシフトさせながら、ゼロ、符号ビット、または加算器の桁上げフラグ(CUおよびCL)を回転または挿入することができる。

【0085】典型的なシフト/回転演算の場合、バレルシフタ470は、両方のソースオペランドの位置にある各ビットを演算によって示される方向に移動させる。各位置のシフトに対して、バレルシフタ470は、選択される演算に依って、終わりのビットを回転させるか、または符号ビット、桁上げフラグ(CUまたはCL)、もしくはゼロを挿入する。

【0086】たとえば、左回転の場合、ビットは左側へシフトされる。ビット31はワードモードではビット0にシフトされる。ハーフワード対モードの場合には、ビット31はビット16に回転させられ、ビット15はビット0に回転させられる。右回転の場合は、ビットは右側にシフトされる。ゼロはワードモードではビット31に挿入される。ハーフワード対モードの場合には、ゼロはビット31およびビット15の両方に挿入される。同様に、キャリー伝搬を伴うシフトでは、桁上げフラグ

(CU)はワードモードではビット31に挿入される。ハーフワード対モードの場合には、各ハーフワードの桁上げフラグ(CUおよびCL)はビット31およびビット15に挿入される。

【0087】次に図15を参照する。デュアルMACユニットは、2つの16×16の積または16×32の積のいずれをも生ずることができるよう一体的に相互接続された、2つのMACユニット520、550、570、590および510、540、560、580から

構成される。各MACは、 16×16 乗算アレイ510、520と、累算加算器560、570と、累算器レジスタファイル580、590と、スケラ591とから構成される。

【0088】幾つかの例示的な命令：乗算、累算、乗算および累算、ユニバーサルハーフワード対乗算、ユニバーサルハーフワード対乗算および累算、ダブル乗算ステップ、ならびにダブル乗算および累算ステップが、図28-図36に挙げられる命令のまとめに見られる。

【0089】ワード演算はどちらかのMACユニットで実行され得る。MACは現在 16×16 演算であるため、MACユニットで用いられる「ワード」は16ビットであることは注目されるべきである。しかしながら、より便利なアプローチは、ベクトル長1、2、4または8を用いて演算を表わすことである。したがって、MACにおけるワード演算はベクトル長1と呼ばれることができ、一方ハーフワード対演算はベクトル長2となるだろう。宛先累算器を含むMACは、演算を行なうのに現在用いられているものである。

【0090】ハーフワード対演算は両方のMACユニットを用いる。命令は特定の累算器を宛先累算器として特定し、これはアドレス指定される累算器となる。アドレス指定される宛先累算器を含むMACは下位のハーフワード対要素で演算を行ない、他方の（「対応する」）MACは同じ演算を上位のハーフワード対要素で行なう。対応するMACからの結果は対応する累算器にストアされ、アドレス指定される累算器と対応する累算器とはそれらのそれぞれのレジスタファイルにおいて同じ相対位置に位置する。

【0091】倍精度演算はハーフワードおよびワードで行なわれ、この演算は二重MACとして組合せられる2つのMACによって行なわれる。「上位」MACは計算の最上位部を行ない、「下位」MACは計算の最下位部を行なう。

【0092】MACユニットは、整数オペランドまたは小数オペランド、および符号付または符号なしオペランドをサポートしてもよい。

【0093】累算器レジスタファイル

2つのMACユニットは上位MACおよび下位MACと呼ばれる。各MACは4つの40ビットのガードされる累算器レジスタから構成される累算器レジスタファイルを有し、ALUには合計8つの累算器がある。各ガードされる累算器（AGn）は、最上位端が8ビットのガードレジスタ（Gn）でもって拡張される32ビット累算器レジスタ（An）から構成される。図16は累算器レジスタファイルのレイアウトを示す。

【0094】ハーフワード対オペランドの累算器は2つの累算器にストアされる。ハーフワード対の下位要素は、いずれかのMACの1つの累算器において、40ビット数として累算される。ハーフワード対の上位要素

は、他方のMACにある対応する累算器において、40ビット数として累算される（図17は対応するアドレスを示す）。

【0095】2つの累算器は、倍精度ステップ演算の結果をストアするためにさらに用いられる。結果の最上位部は、上位MACのガードされる累算器AGにストアされる。結果の最下位部は、下位MACの累算器Aにストアされる。下位MAC累算器のガードビットは使用されない。

【0096】各累算器は、レジスタ空間に、上位および下位累算器アドレスまたは上位および下位冗長アドレスと呼ばれる2つのアドレスを有する。（累算器nのためのこれらのアドレスのアセンブリ言語名はそれぞれAnHおよびAnLである。）どちらのアドレスが使用されるかというこの効果は、レジスタが命令においてどのように用いられるかに依存し、これらの効果は以下のサブセクションにおいて詳細に述べられる。

【0097】命令フォーマット（およびアセンブリ言語）はアドレス指定累算器の幾つかの方法を提供する。

【0098】・レジスタ空間の要素として。各累算器は、111ないし127の範囲に、アセンブリ言語記号をARnHおよびARnLとする上位アドレスおよび下位アドレスを有する。

【0099】・累算器オペランドとして。命令フォーマットは範囲0-7にある数を取り、対応するアセンブリ言語記号はAn形式である。

【0100】・別々の上位アドレスおよび下位アドレスを有する累算器オペランドとして。命令フィールドは範囲0-15にある値を取り、アセンブリ言語フォーマットはAnHまたはAnLである。

【0101】8つのガードレジスタの各々は拡張レジスタ空間にアドレスを有する（160-167；アセンブリ言語記号はAGn形式を有する）。

【0102】このセクションの残りのサブセクションは、累算器およびガードレジスタの、命令としての取扱いを特定する。レジスタがソースであるかまたは宛先であるか、および演算の要素がワードであるかまたはハーフワード対であるかによって、多数の特別な例がある。

【0103】1. ワードソースオペランドとしての累算器

上位累算器アドレスは累算器Anの上位32ビットを小数ワードオペランドとして特定し、下位アドレスはAnの下位32ビットを整数ワードオペランドとして特定する。プロセッサの現在のバージョンでは、累算器は32ビットの長さなので、両方のアドレスとも同じ32ビットを参照する。しかしながら、一般的なプロセッサアーキテクチャはより長い累算器を可能にする。ガードビットは、累算器（アセンブリ言語An）を32ビットソースオペランドとして用いる命令によって無視される。命令が、ガードされる累算器（アセンブリ言語AGn）を

用いることを、たとえば累算レジスタのためにまたはスケラへの入力として特定する場合には、ガードビットは40ビットソースオペランドに含まれる。

【0104】バス構造は、現在、各MACからの1つの累算器レジスタが任意の所与の命令において明示されるソースオペランドとして用いられることを可能にする。

【0105】累算器が乗算演算のためのソースオペランドとして選択されると、32ビットすべてが累算器によって提示される。命令はさらに、整数/小数オプションによって、乗算アレイへの入力のための下位または上位ハーフワードを選択する。

【0106】2. ハーフワード対ソースオペランドとしての累算器

ハーフワード対の各要素は、累算器に、あたかもワードオペランドであるかのように保持される。ハーフワード対の2つの要素は、別個のMACにある対応する累算器にストアされる。それらのそれぞれのMAC内で累算器レジスタとしてまたはスケラへの入力として用いられるときは、それらは40ビットソースオペランドとして用いられる。

【0107】それ以外の場合には、要素はハーフワード対オペランドで2つのハーフワードとしてアセンブルされる。ハーフワード対ソースオペランドが上位累算器アドレスである場合には、各要素に対し累算器の上位ハーフワードが用いられる。下位累算器アドレスが用いられる場合には、下位ハーフワードが用いられる。アドレス指定される累算器は下位ハーフワードを与え、対応する累算器は上位ハーフワードを与える。いずれのMACもハーフワード対のいずれの要素をも供給することができる。

【0108】3. 倍精度ソースオペランドとしての累算器

累算器は倍精度ステップ演算においてのみ精度ソースオペランドのために用いられる。アドレス指定される累算器は最下位32ビットを与え、対応するガードされる累算器は最上位40ビットを与える。

【0109】4. ソースオペランドとしてのガードレジスタ

8ビットガードレジスタ(Gx)は符号拡張整数として拡張レジスタ空間から直接アクセスすることができる。ガードレジスタがハーフワード対演算のソースオペランドである場合、アドレス指定されるガードは最下位ハーフワードオペランドとなり、対応するガードは最上位ハーフワードオペランドとなる。両方の例において、ガードレジスタは16ビットに符号拡張される。

【0110】5. ワード宛先オペランドとしての累算器
MACを用いるワード演算では、乗算演算の32ビット結果は、宛先累算器にストアされ、そのガードレジスタを介して符号拡張される。累算演算の40ビット結果は宛先ガード累算器にストアされる。

【0111】他の、レジスタからレジスタへの命令では、結果は、宛先累算器に移動させられ、そのガードレジスタを介して符号拡張される。

【0112】6. ワード対宛先オペランドとしての累算器

ワード対のデータタイプの変換を特定する累算器を目標とするロード命令では、下位メモリアドレスからのワードはアドレス指定される累算器にロードされ、より上位のメモリアドレスからのワードの最下位バイトは累算器のガードレジスタにロードされる。

【0113】7. ハーフワード対宛先オペランドとしての累算器

2つのMACユニットを用いるハーフワード対演算では、各MACの結果はその累算器ファイルにストアされる。宛先累算器を含むMACは下位のハーフワード対要素を処理し、その40ビット結果はそのガードされる累算器(AG)にストアされる。対応するMACは上位のハーフワード対要素を処理し、その40ビット結果は対応するガードされる累算器(AGC)にストアされる。

【0114】他の、レジスタからレジスタへの命令では、宛先累算器のために選択される特定の累算器アドレスが、結果をどのようにストアするかを判断する。上位アドレスが用いられる場合には、最下位ハーフワードは、選択される累算器の最上位半分にロードされ、右側へゼロ拡張され、そのガードレジスタを介して符号拡張される。最上位ハーフワードは、対応する累算器の最上位半分にロードされ、右側へゼロ拡張され、そのガードレジスタを介して符号拡張される。下位アドレスが用いられる場合には、最下位ハーフワードは、選択される累算器の最下位半分にロードされ、選択される累算器の最上位半分を介し、次いでそのガードレジスタを介して符号拡張される。最上位ハーフワードは、対応する累算器の最下位半分にロードされ、上述のように符号拡張される。

【0115】8. 倍精度オペランドとしての累算器

倍精度乗算ステップ演算の結果の最下位32ビットは宛先累算器にストアされ、最上位40ビットは対応するガードされる累算器にストアされる。宛先累算器のガードビットはすべてゼロにセットされる。

【0116】9. 宛先オペランドとしてのガードレジスタ

ガードレジスタが宛先オペランドである場合、結果の8つの最下位ビットはアドレス指定されるガードレジスタにストアされる。ガードレジスタがハーフワード対演算の宛先オペランドとしてもちいられる場合には、結果の8つの最下位ビットはアドレス指定されるガードレジスタにストアされ、上位ハーフワードの8つの最下位ビットは対応するガードレジスタにストアされる。

【0117】乗算アレイ

ここで図15を参照する。各MACのための乗算アレイ

または乗算ユニットは、2つの16ビット入力から32ビットの積を生ずる。符号付および符号なし入力、整数および小数入力は、任意の組合せで乗算されてもよい。整数入力の場合、ソースオペランドの最下位ハーフワードが用いられる。小数入力の場合は、最上位ハーフワードが用いられる。図18は入力のスケーリングを示し、図19は出力スケーリングを示す。

【0118】2つのワードオペランドまたは1つのワードおよび1つの即値オペランドが乗算される場合には、宛先累算器を含むMACのみが用いられる。2つのHPオペランドまたは1つのHPおよび1つの即値オペランドが乗算される場合には、両方のMACが用いられ、宛先累算器を含むMACは下位のHP要素を乗算する。

【0119】ともに用いられる2つの乗算アレイは、図18に従ってスケーリングされる1つの16ビット入力と1つの32ビット入力とから48ビットの積を生ずる。この積は、図20および図21に従ってスケーリングされる。

【0120】乗算飽和

-1.0が累算なしで(16ビットの符号付小数として)-1.0によって乗算される場合、結果(+1.0)は飽和して、ガードビットへのオーバフローを防ぐ。最大の正の数は累算器(A)に置かれ、ガードビットはゼロにセットされる。乗算命令が累算を含む場合には、結果は飽和せず、代わりに完全な結果が宛先ガード累算器において累算されそこに置かれる。

【0121】乗算スケーリング

図18、図19、図20および図21は、乗算演算のためのソースオペランドおよび結果のスケーリングを示す。表は、小数点の想定された位置および任意の符号ビットの処理を示す。

【0122】図18は乗算演算のためのソースオペランドのスケーリングを示す。図19は32ビットの積のためのスケーリングを示す。図20および図21は48ビットの積のためのスケーリングを示す。(図20(a)および(b)は、下位および上位MACにおいてそれぞれ右寄せされる積のスケーリングを示し、同様に図21(a)および(b)は左寄せされた積のスケーリングを示す。)

累算加算器

図15を参照すると、各MACは、累算器に入力を加算することのできる(または累算器から入力を減算することのできる)累算加算器を含む。考えられ得る入力は、乗算アレイからの積、即値オペランド、いずれかのMACからの累算器、またはワードもしくはハーフワード対を含むレジスタである。

【0123】累算初期化特性は、ステータスレジスタ(ST)(図示せず)のIMAC(抑止MAC累算)ビットによって制御される。乗算/累算演算を行なう命令が実行され、IMACビットが真(=1)である場合に

は、宛先累算器は入力オペランドに初期化され、IMACビットは偽(=0)にリセットされる(実際には、宛先累算器は、入力が累算される前に0にセットされる)。

【0124】同様の初期化および丸め特性は、ステータスレジスタのIMARビットによって制御される。IMARビットが真である間に、累算加算器演算を行なう命令が実行されると、累算レジスタは丸め係数によって置き換えられ、宛先累算器は入力オペランドに切上げビットを加えたものに初期化され、IMARビットは偽にリセットされる。丸め係数は、下位ハーフワードの最上位ビットにある1を除き、すべて0である。

【0125】いくつかの乗算命令は、累算加算器において実行される丸めオプションを含む。丸められた結果は宛先累算器の上位ハーフワードに置かれ、ゼロは下位ハーフワードに置かれる。結果は、下位ハーフワードと上位ハーフワードとの間に小数点を有すると考えられるべきであり、結果は最も近い整数に丸められ、下位ハーフワードが1/2である場合には(つまり上位ビットが1である場合には)、結果は最も近い偶数の整数に丸められる。

【0126】累算加算器のオーバフローはオーバフローフラグをセットしない。飽和オプションを有する累算命令に対してオーバフロー生ずると、ガードされる累算器はオーバフローの方向に従ってその最も大きい正の数または最も小さい負の数にセットされる。命令が飽和を特定せず、かつ誤り例外が可能化される場合には、オーバフローは誤り例外要求を生ずる。飽和を有するオーバフローおよび飽和を有しないオーバフローのために、デバッグ論理に別個の信号が送られる。

【0127】図22は、累算レジスタに加算されるワードまたは累算器オペランドを示す。図23は、累算レジスタにあるハーフワード対に加算される(レジスタまたは累算器からの)ハーフワード対オペランドを示す。

【0128】図24は、累算レジスタに加算される積を示す。図25は、累算レジスタにあるハーフワード対に加算されるハーフワード対の積を示す。

【0129】図26は、右寄せオプションを用いて累算される48ビットの積を示す。このオプションは、整数結果が所望される 16×32 積、または 32×32 積の第1のステップに適用できる。

【0130】図27は、左寄せオプションを用いて累算される48ビット積を示す。このオプションは、小数結果が所望される 16×32 積、または 32×32 積の第2のステップに適用できる。

【0131】図28-図36は、この発明の空間ベクトルデータ経路に従って実現されるであろう演算の命令のまとめである。

【0132】スケール

図15を参照すると、スケールユニットは、ガードされ

る累算器の全長上で、0ないし8ビット位置の右バレルシフトを行なうことができる。最上位ガードビットは空いたビットに伝搬される。

【0133】ガードビットと結果の最上位ビットとがすべて一致しない場合には、スケーラ命令の間にオーバフローが生ずる。(これらのビットが一致する場合には、それは、累算器の符号ビットがガードレジスタ全体を通して伝搬し、累算器のオーバフローはガードビットには生じなかったことを意味する。)

スケーラ命令はオーバフローが生じた際に結果を飽和するオプションをサポートする。この例においては、結果は、オーバフローの方向に依って、最も大きい正の数または最も小さい負の数に1つの最下位ビットを加えてものにセットされる(最上位ガードビットは、元の数が正であったかまたは負であったかを示す。)

オーバフローが生じ、飽和が特定されなかったときに、誤り例外が可能化された場合には誤り例外が生ずる。飽和のないオーバフローおよび飽和のあるオーバフローは、別個の信号でデバッグ論理に報告される。

【0134】累算器を正規化するのに、レジスタへの移動がスケーリングされる累算器(MAR)を用いてもよい。累算器Anを正規化するためには:

MAR Rx, AnH, #8; 8ビットでAGnをスケーリングする

MEXP Rc, Rx; 指数を測定する

SUBRU, W, SAT Rc, Rc, #8; 正規化に必要なシフト数を計算する

MAR Rx, AnH, Rc; 累算器の内容を正規化する

このシーケンスの後、Rcはガードされる累算器を正規化するのに必要なシフト数を含み、Rxは正規化された結果を含む。

【0135】この発明は図1-図36を参照して記載されてきたが、この発明の教示は当業者によって決定されるようなさまざまな処理スキームに適用されてもよいことが理解される。

【図面の簡単な説明】

【図1】(a)は、従来の単一命令、多重データ(SIMD)コンピュータの概念的な図である。(b)はSIMDコンピュータに用いられる処理素子の単純な図である。

【図2】この発明を組み込むであろうプログラマブルプロセッサの一般化された図である。

【図3】(a)は、処理ユニットのためのALUに組み込まれるであろう従来の加算器の模式図である。(b)および(c)は、この発明を実現するであろう加算器の模式図である。

【図4】(a)は、処理ユニットのためのALUに組み込まれるであろう従来の論理ユニットの模式図である。

(b)および(c)は、この発明を実現するであろう論

理ユニットの模式図である。

【図5】(a)および(b)は、この発明を実連するであろう従来のシフトの模式図である。

【図6】この発明を組み込むであろうシフトの図である。

【図7】この発明を組み込むであろうシフトの図である。

【図8】この発明を組み込むであろうシフトの図である。

【図9】従来の乗算累算器(MAC)の単純な図である。

【図10】MACがこの発明をどのように組み込み得るかを示す図である。

【図11】MACが32×16モードでこの発明をどのようにして組み込み得るかを示す図である。

【図12】32×16モードのためのMAC内の相互接続を示す図である。

【図13】この発明を組み込む処理素子の単純な機能図である。

【図14】この発明を組み込むALUおよびシフトの単純な図である。

【図15】デュアルMAC構成を示す図である。

【図16】累算器レジスタファイルのレイアウトを示す図である。

【図17】(a)および(b)は、対応する累算器アドレスを示す図である。

【図18】乗算演算のためのソースオペランドおよび結果のスケーリングを示す図である。

【図19】乗算演算のためのソースオペランドおよび結果のスケーリングを示す図である。

【図20】(a)および(b)は、乗算演算のためのソースオペランドおよび結果のスケーリングを示す図である。

【図21】(a)および(b)は、乗算演算のためのソースオペランドおよび結果のスケーリングを示す図である。

【図22】累算器レジスタに加算されるワードまたは累算器オペランドを示す図である。

【図23】累算レジスタでハーフワード対に加算されるハーフワード対オペランドを示す図である。

【図24】累算レジスタに加算される積を示す図である。

【図25】累算レジスタでハーフワード対に加算されるハーフワード対の積を示す図である。

【図26】右寄せオプションを用いて累算される48ビットの積を示す図である。

【図27】左寄せオプションを用いて累算される48ビットの積を示す図である。

【図28】この発明に従って実現されるであろう命令のまとめを示す図である。

【図29】この発明に従って実現されるであろう命令のまとめを示す図である。

【図30】この発明に従って実現されるであろう命令の

まとめを示す図である。

【図31】この発明に従って実現されるであろう命令のまとめを示す図である。

【図32】この発明に従って実現されるであろう命令のまとめを示す図である。

【図33】この発明に従って実現されるであろう命令のまとめを示す図である。

【図34】この発明に従って実現されるであろう命令のまとめを示す図である。

【図35】この発明に従って実現されるであろう命令のまとめを示す図である。

【図36】この発明に従って実現されるであろう命令のまとめを示す図である。

【符号の説明】

100 プログラムおよびデータ記憶ユニット

110 処理ユニット

121 ALU

122 MAC

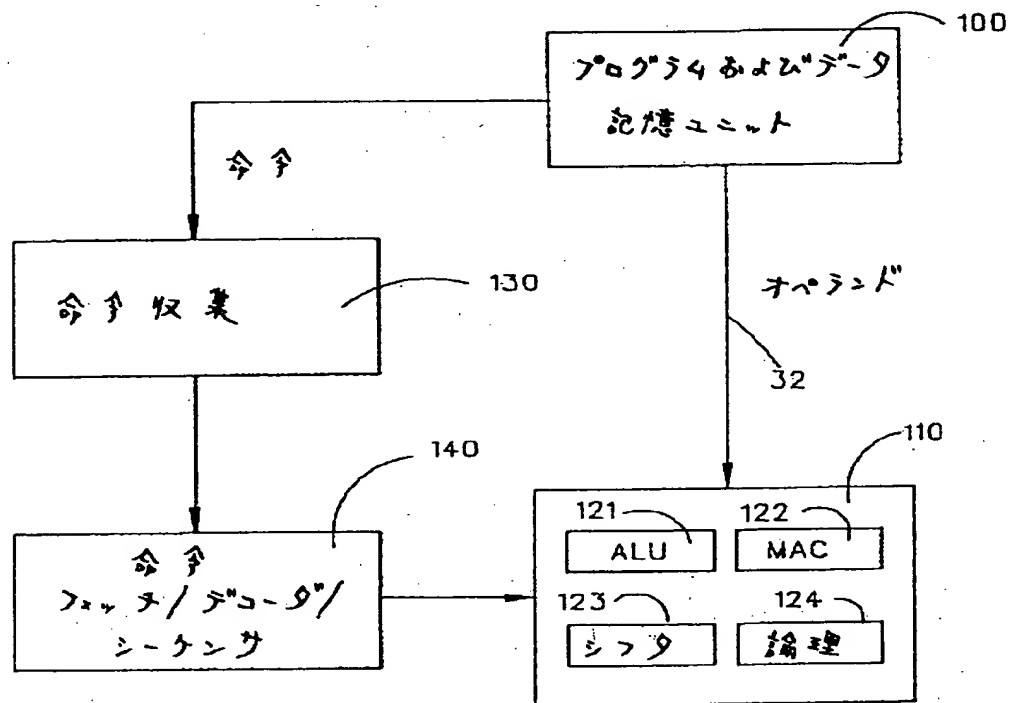
123 シフト

124 論理ユニット

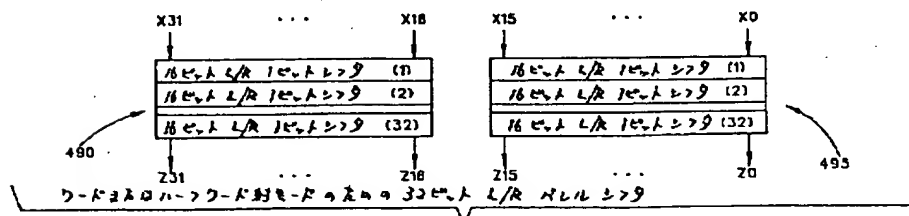
130 命令収集ユニット

140 命令フェッチ/デコーダ/シーケンスユニット

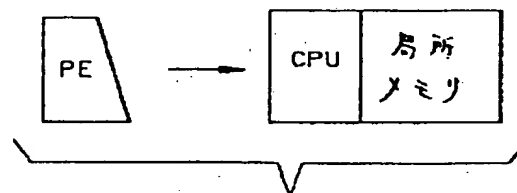
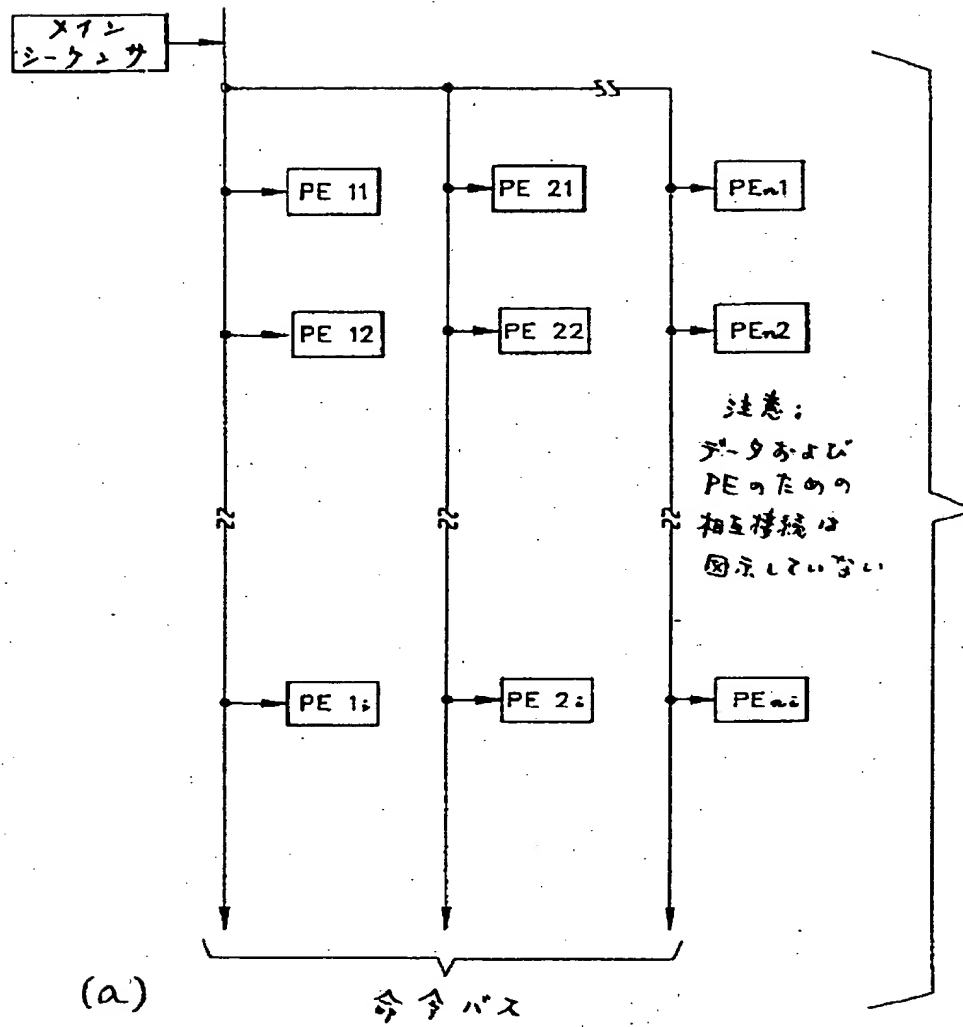
【図2】



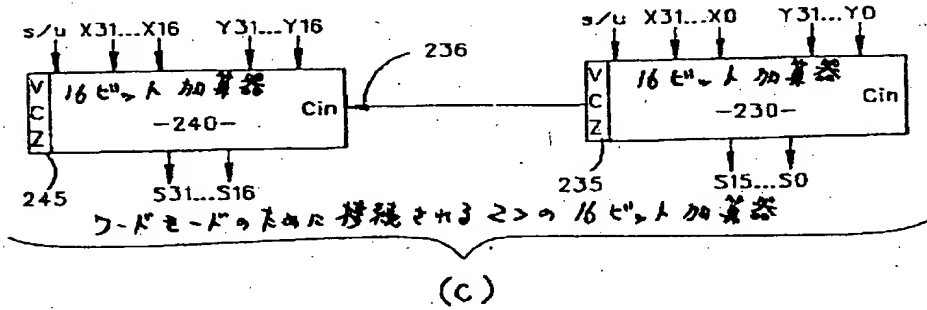
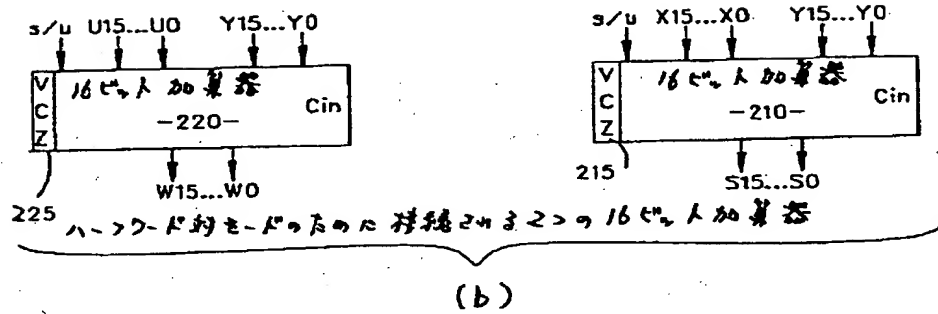
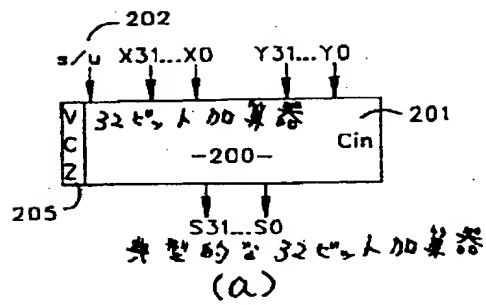
【図8】



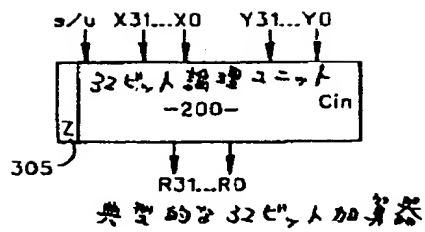
【図 1】



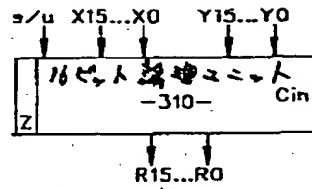
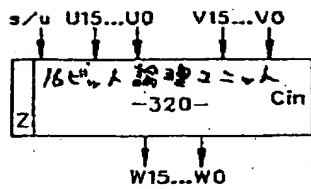
【図3】



【図4】

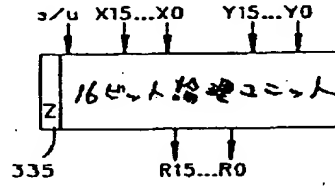
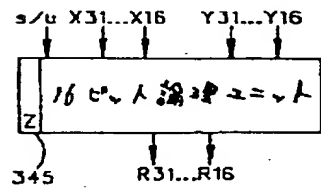


(a)



ハーフワードモードのときに接続される2つの16ビット処理ユニット

(b)



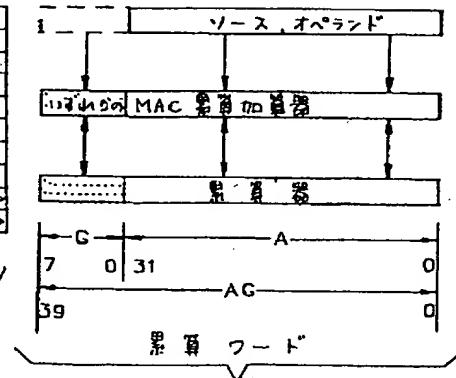
ワードモードのときに接続される2つの16ビット処理ユニット

(c)

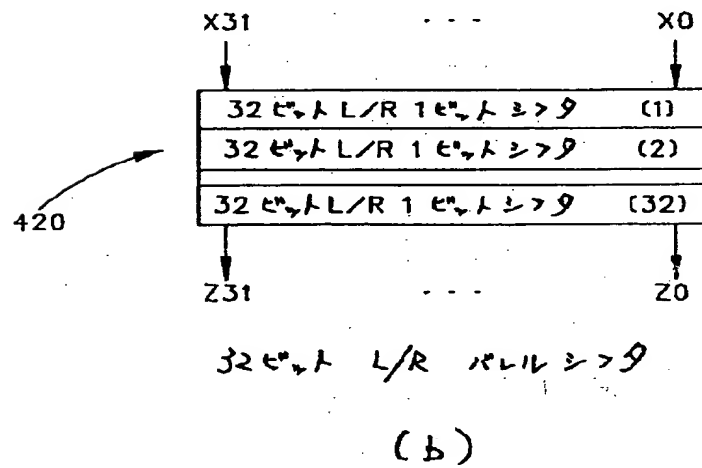
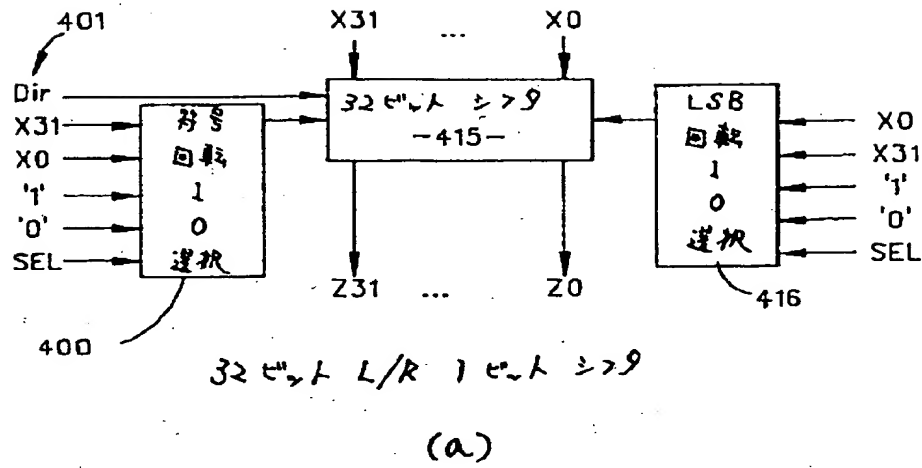
【図16】

下位 MAC 累算器 レジスタファイル	G0	A0
	G1	A1
	G2	A2
	G3	A3
上位 MAC 累算器 レジスタファイル	G4	A4
	G5	A5
	G6	A6
	G7	A7
<Gx(8ビット)>		<Ax (32ビット)>
		<AGx(40ビット)>
累算器 レジスタ		

【図22】



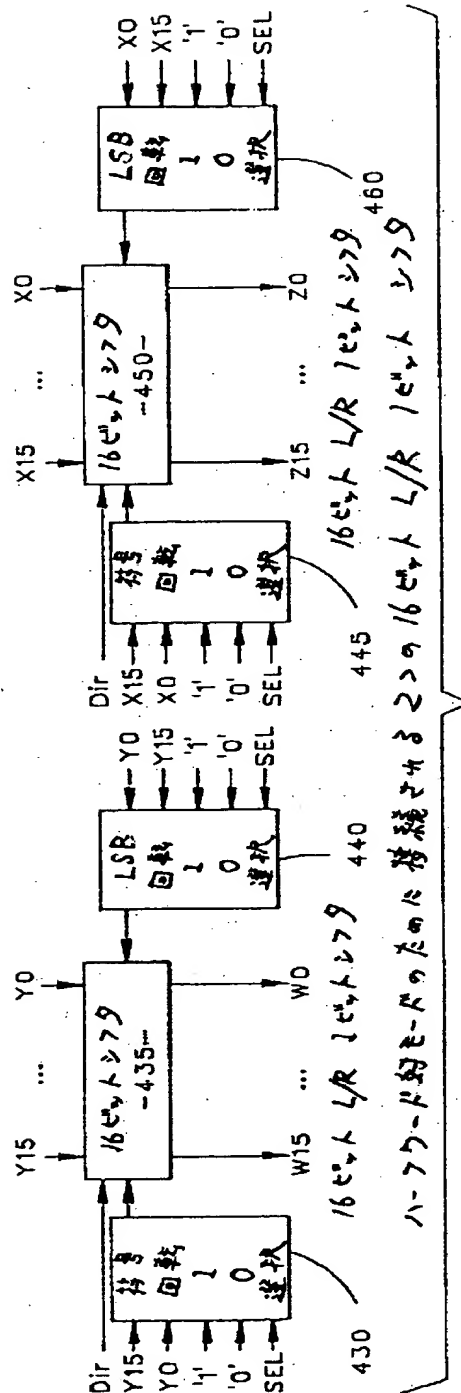
【図5】



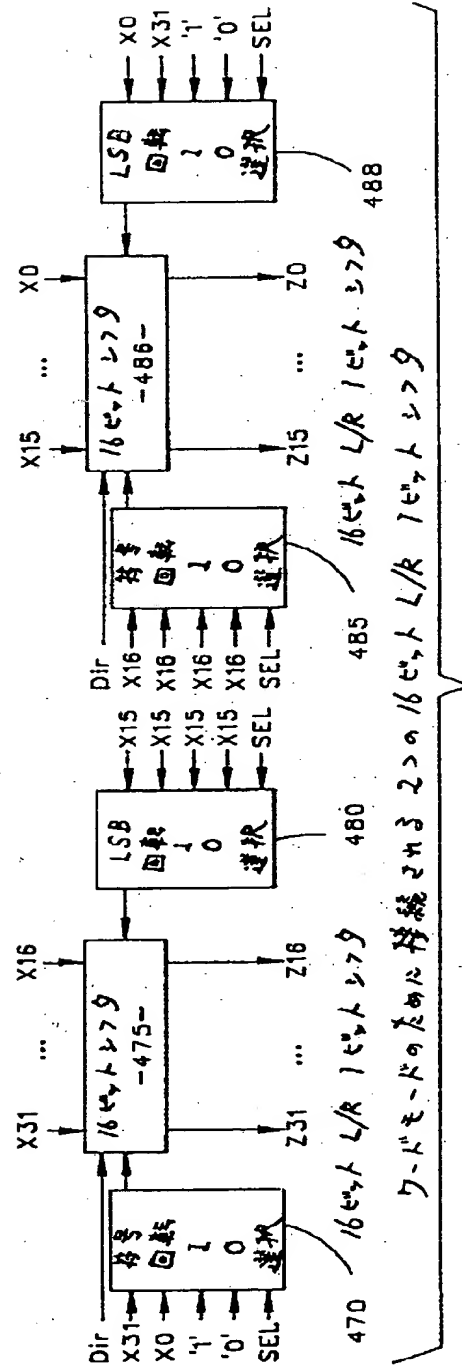
【図18】

乗算器ソースオペランドフォーマット	
タイプ	16ビットソースオペランドフォーマット
SI	S u u u u u u u u u u u u u u u u
UI	u u u u u u u u u u u u u u u u
SP	S u u u u u u u u u u u u u u u u
UP	u u u u u u u u u u u u u u u u
タイプ	32ビットソースオペランドフォーマット
SI	S u
UI	u u
SP	S u
UP	u u
SI 符号付整数 SP 符号付小数 S 符号ビット UI 符号なし整数 UP 符号なし小数 u 符号なしビット	

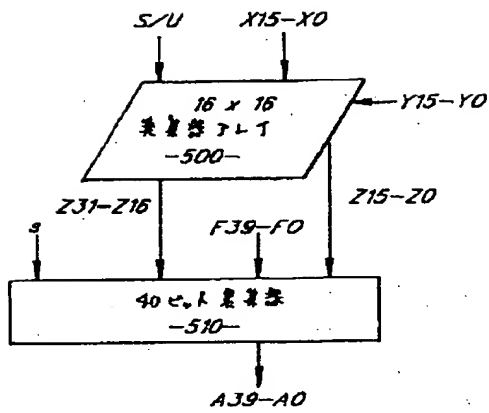
【図6】



【図7】

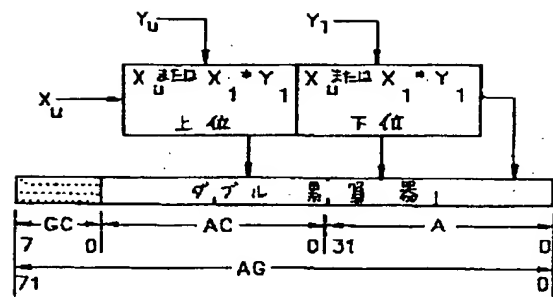


【図 9】

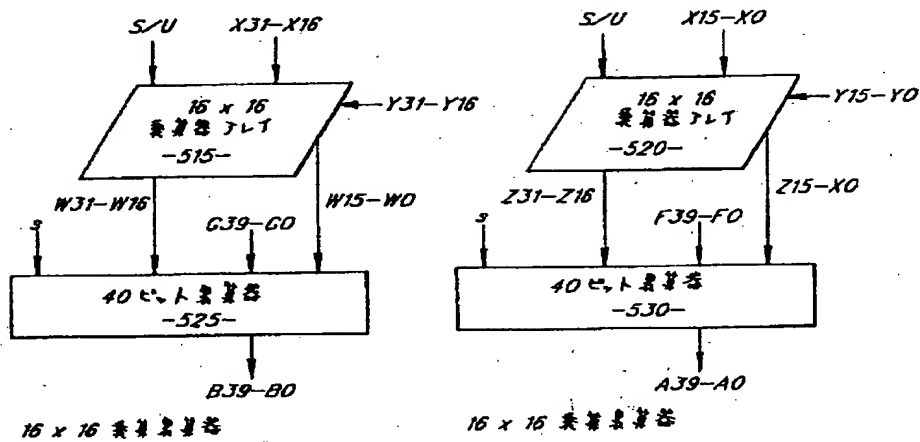


従来の16x16乗算器

【図 26】



【図 10】

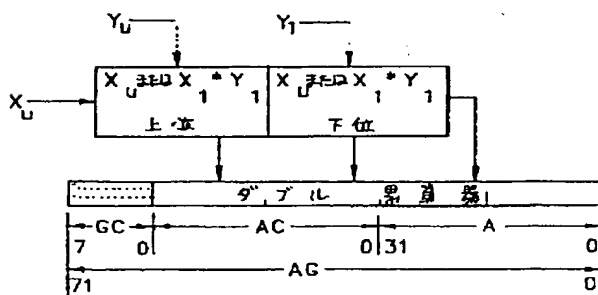


16x16 乗算器

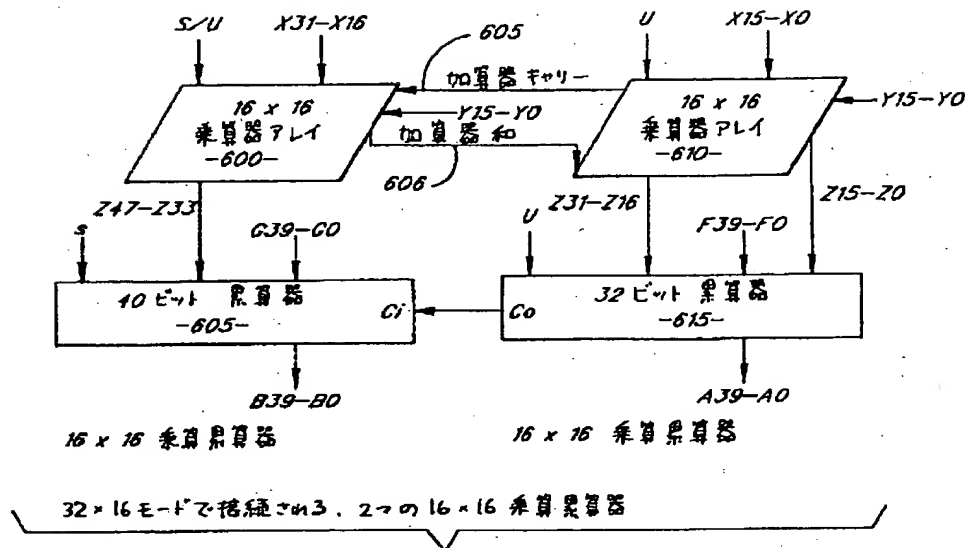
16x16 乗算器

ハーフワードモードでの2つの16x16乗算器

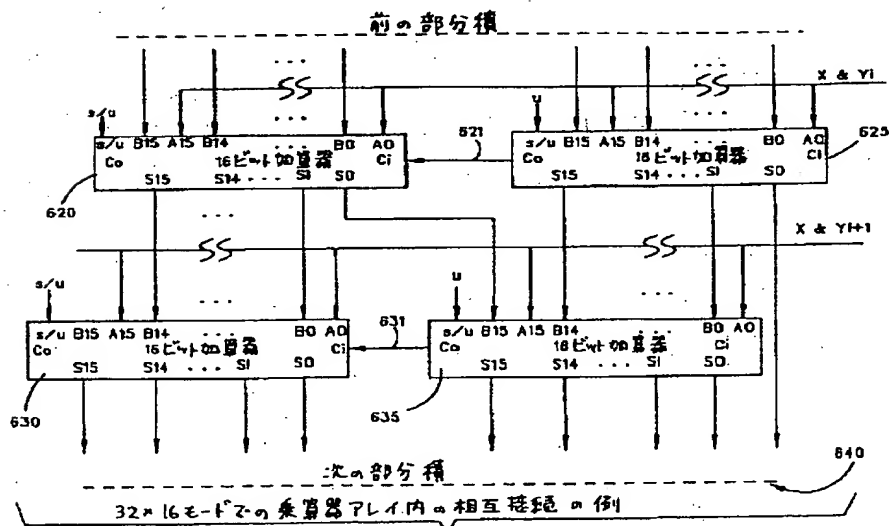
【図 27】



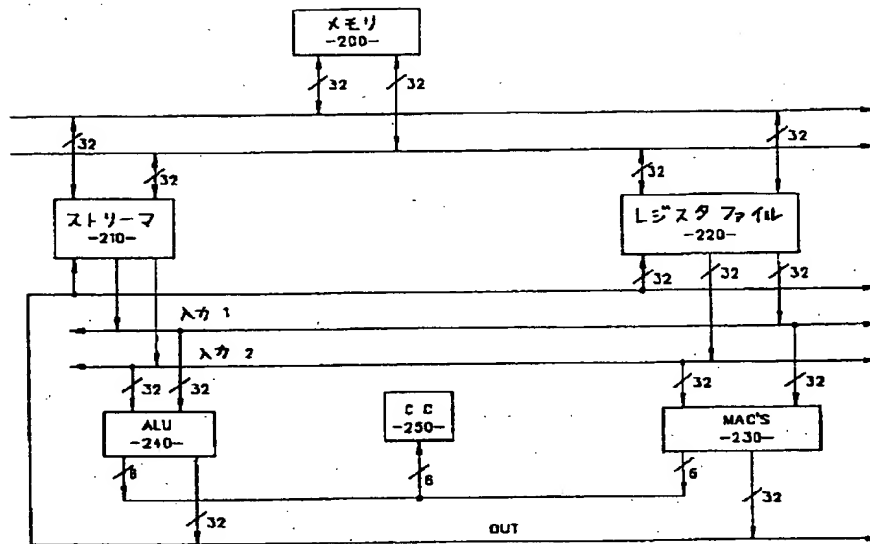
【図 11】



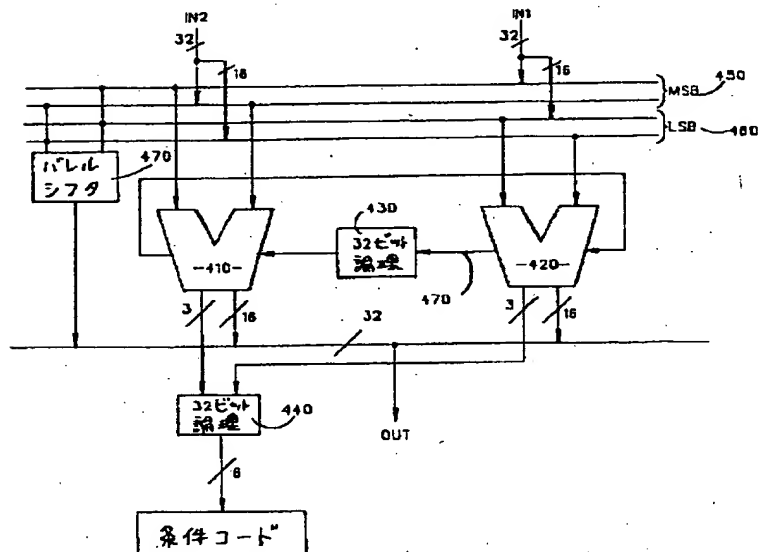
【図 12】



【図13】



【図14】



【図17】

x	AGx	AGCx	Ax	ACx
0	AG0	AGC4	A0	A4
1	AG1	AGC5	A1	A5
2	AG2	AGC6	A2	A6
3	AG3	AGC7	A3	A7
4	AG4	AGC0	A4	A0
5	AG5	AGC1	A5	A1
6	AG6	AGC2	A6	A2
7	AG7	AGC3	A7	A3

対応する累算器アドレス指定

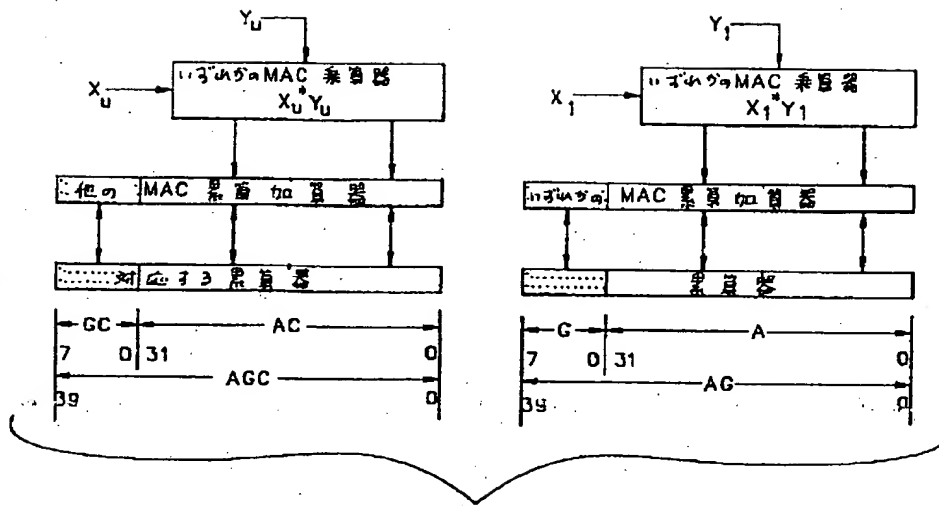
(a)

SI	符号付整数
UI	符号なし整数
SF	符号付小数
UF	符号なし小数
S	符号ビット
U	符号なしビット
S	符号拡張ビット
O	ゼロ拡張ビット
G	ガードビット
A	累算ビット

図18-図21(b)の記号

(b)

【図 25】



【图 28】

【图 29】

RSP™ 命令のまじり 一動作順序

[illegible]

ニームニツク アセンブラシタックス 創作

[illegible]

【図30】

ニーモニック	アセンブラシンタックス	動作
LL	LLS dest, #imm_16 LLU dest, #imm_16	レジスタに符号付即時値をロードする レジスタに符号無し即時値をロードする
LLF	LLF dest, #imm_16	レジスタに即時値 0 (符号付) をロードする
LLFD	LLFD dest, #imm_16	レジスタに即時値 0 をロードしてマージする (符号付)
LLM	LLM dest, #imm_16	レジスタに即時値をロードしてマージする (符号付無符号)
LLD	LLD dest, #imm_16	レジスタに符号付即時値をロードする (符号付無符号)
LLUD	LLUD dest, #imm_16	レジスタに符号付即時値をロードする (符号付無符号)
LLFD	LLFD dest, #imm_16	レジスタに符号付即時値 0 をロードする (符号付)
LLMD	LLMD dest, #imm_16	レジスタに符号付即時値をロードしてマージする (符号付)
ML	MLW MLB	ワードを移動する ハーフワードを移動する (MLNP に移動する)
MCC	MCC, HP dest, src MCC, W, <cc> dest, src	Cビットがクリアとされた後移動する
MCS	MCS, HP dest, src MCS, W, <cc> dest, src	Cビットがセットされた後移動する
MVC	MVC, HP dest, src MVC, W, <cc> dest, src	Vビットがクリアとされた後移動する
MVS	MVS, HP dest, src MVS, W, <cc> dest, src	Vビットがセットされた後移動する
MZC	MZC, HP dest, src MZC, W, <cc> dest, src	Zビットがクリアとされた後移動する
MZS	MZS, HP dest, src MZS, W, <cc> dest, src	Zビットがセットされた後移動する
MZ	MZ, HP dest, src, 1, src_2 MZ, W, <cc> dest, src, 1, src_2	ゼロに等しい後移動する
MONZ	MONZ, HP dest, src, 1, src_2 MONZ, W, <cc> dest, src, 1, src_2	ゼロに等しくゼロの後移動する
MLT	MLT, HP dest, src, 1, src_2 MLT, W, <cc> dest, src, 1, src_2	より小さければ移動する
MGT	MGT, W, <cc> dest, src, 1, src_2	より大きければ移動する
MLE	MLE, HP dest, src, 1, src_2	より小さいかまたは等しいの後移動する
MCE	MCE, HP dest, src, 1, src_2 MCE, W, <cc> dest, src, 1, src_2	より大きい、または等しいの後移動する

【図31】

ニーモニック	アセンブラシンタックス	動作
MBZ	MBZ dest, src, 1, src_2, #imm_16, imm	ゼロ上のビットを移動する
MBNZ	MBNZ dest, src, 1, src_2, #imm_16, imm	ゼロ上でゼロのビットを移動する
MRA	MRA, B dest, src MRA, H dest, src MRA, HP dest, src MRA, W dest, src	レジスタから累算器へ移動する -ハーフワード レジスタから累算器へ移動する -ハーフワード レジスタから累算器へ移動する -ハーフワード レジスタから累算器へ移動する -ワード
NAR	NAR dest, src, 1, src_2	ステリングされた累算器のレジスタへ移動する
PK	PK, H, L dest, src, 1, src_2 PK, H, L, H dest, src, 1, src_2 PK, H, L, H dest, src, 1, src_2 PK, B, P, L dest, src, 1, src_2 PK, B, P, H dest, src, 1, src_2	下位ハーフワードを上位にパックする 上位ハーフワードを上位にパックする 上位ハーフワードを上位にパックする 下位バイトを上位にパックする PK, B, P, L に移動する 上位バイトを上位にパックする
B	B, <cc> 16	条件付き
BCC	BCC, <cc> 12	Cビットがクリアとされた後移動する
BCS	BCS, <cc> 12	Cビットがセットされた後移動する
BVC	BVC, <cc> 12	Vビットがクリアとされた後移動する
BVS	BVS, <cc> 12	Vビットがセットされた後移動する
BZC	BZC, <cc> 12	Zビットがクリアとされた後移動する
BZS	BZS, <cc> 12	Zビットがセットされた後移動する
BZ	BZ, <cc> 12	レジスタがゼロに等しいの後移動する
BNZ	BNZ, <cc> 12	レジスタがゼロに等しくないの後移動する
BLT	BLT, <cc> 12	レジスタがゼロより小さいの後移動する
BGE	BGE, <cc> 12	レジスタがゼロより大きいかまたは等しいの後移動する
BGT	BGT, <cc> 12	レジスタがゼロより大きければ移動する
BLE	BLE, <cc> 12	レジスタがゼロより小さいかまたは等しいの後移動する

【図32】

ニーモニック	アセンブラシンタックス	動作
BZ<cd>	BZ reg, #bit_mask, of<cd>	ゼロに等しいビット上で分岐する
BNZ<cd>	BNZ reg, #bit_mask, of<cd>	ゼロに等しくないビット上で分岐する
BRCZ<cd>	BRCZ reg, #bit_mask, of<cd>	ゼロに等しいビットおよび補数ビット上で分岐する
BRNCZ<cd>	BRNCZ reg, #bit_mask, of<cd>	ゼロに等しくないビットおよび補数ビット上で分岐する
BZQ<cd>	BZQ reg, 1, reg, 2, of<cd>	レジスタ0-1のビット上で分岐する
BNZQ<cd>	BNZQ reg, 1, reg, 2, of<cd>	レジスタ0-1のビット上で分岐する
BNZD<cd>	BNZD reg, #mask, of<cd>	ゼロに等しくないビット上で分岐する
BNZL<cd>	BNZL reg, #mask, of<cd>	ゼロに等しくないビット上で分岐する
J<cd>	J addr, 22	ジャンプする
J<cd>v<cd>	J (reg)	ジャンプする
JSB<cd>v<cd>	JSB (reg, streamer)	ストリーマにバイトをジャンプする
JSH<cd>v<cd>	JSH (reg, streamer)	ストリーマにワードをジャンプする
JC<cd>v<cd>	JC (reg, 1, reg, 2)	条件付ジャンプする
JCSB<cd>v<cd>	JCSB (streamer, reg)	条件付ストリーマにバイトをジャンプする
JCSH<cd>v<cd>	JCSH (streamer, reg)	条件付ストリーマにワードをジャンプする
CALL<cd>	CALL addr, 22, #reg_count	サブルーチンコールする
TRAP<cd>	TRAP addr, 22	トラップする
TRAPC<cd>	TRAPC addr, 22	条件付トラップ

【図33】

ニーモニック	アセンブラシンタックス	動作
RETE<cd>	RETE	外部にリターン
RETL<cd>	RETL	外部にリターン (RETEと同じ動作を繰り返す)
RETT<cd>	RETT	外部にリターン (RETEと同じ動作を繰り返す)
RETD	RETD	外部にリターン (RETEと同じ動作を繰り返す)
NOP<cd>	NOP	何もしない
LOOP<cd>	LOOP of<cd>, #loop_count	ループ
WAIT	WAIT	待機
HALT	HALT	停止
BREAK	BREAK	中断
ABS<cd>	ABS dest, src	絶対値
NEG<cd>	NEG dest, src	否定 (1の補数)
NOT<cd>	NOT dest, src	1の補数
PAR<cd>	PAR dest, src	パリティ
PARO<cd>	PARO dest, src	パリティ
REV<cd>	REV dest, src	ビット逆転
ADD(S)<cd>	ADD dest, src, 1, src, 2	加算 (符号付)
ADDU<cd>	ADDU dest, src, 1, src, 2	加算 (無符号)
ADDC(S)<cd>	ADDC dest, src, 1, src, 2	加算 (符号付)
ADDCU<cd>	ADDCU dest, src, 1, src, 2	加算 (無符号)
SUB(S)<cd>	SUB dest, src, 1, src, 2	減算 (符号付)
SUBU<cd>	SUBU dest, src, 1, src, 2	減算 (無符号)
SUBC(S)<cd>	SUBC dest, src, 1, src, 2	減算 (符号付)
SUBCU<cd>	SUBCU dest, src, 1, src, 2	減算 (無符号)
SUBR(S)<cd>	SUBR dest, src, 1, src, 2	減算 (符号付)

【図34】

オペランド	アセンブリ記法	動作
SUBRU<op><src>	SUBRU dest,src,1,0imm	符号なし減算1進値
ASCL<op>	ASCL dest,src,1,src,2,0imm	符号なし減算1進値
MIN(S)	MIN dest,src,1,src,2	最小値(符号あり)
MINU	MINU dest,src,1,src,2	符号なし最小値
MAX(S)<op>	MAX dest,src,1,src,2	最大値(符号あり)
MAXU<op>	MAXU dest,src,1,src,2	符号なし最大値
TRQ<op>	TRQ dest,src,1,src,2	0に等しいビットをクリアする
TNE<op>	TNE(S) dest,src,1,src,2 TNE(U) dest,src,1,0imm	0に等しいビットをクリアする
TLT(S)<op>	TLT(S) dest,src,1,src,2 TLT(U) dest,src,1,0imm	0より小さいビットをクリアする
TGT(S)<op>	TGT(S) dest,src,1,src,2 TGT(U) dest,src,1,0imm	0より大きいビットをクリアする
TLT(S)<op>	TLT(S) dest,src,1,src,2 TLT(U) dest,src,1,0imm	0より小さいビットをクリアする
TGT(S)<op>	TGT(S) dest,src,1,src,2 TGT(U) dest,src,1,0imm	0より大きいビットをクリアする
TLTU<op>	TLTU dest,src,1,src,2 TLTU dest,src,1,0imm	符号なし0より小さいビットをクリアする
TGTU<op>	TGTU dest,src,1,src,2 TGTU dest,src,1,0imm	符号なし0より大きいビットをクリアする
TLEU<op>	TLEU dest,src,1,src,2 TLEU dest,src,1,0imm	符号なし0より小さいビットをクリアする
TGEU<op>	TGEU dest,src,1,src,2 TGEU dest,src,1,0imm	符号なし0より大きいビットをクリアする
TAND<op>	TAND dest,src,1,src,2 TAND dest,src,1,0imm	ビット単位ANDの結果をクリアする
TOR<op>	TOR dest,src,1,src,2 TOR dest,src,1,0imm	ビット単位ORの結果をクリアする
SBIT<op>	SBIT dest,src,1,src,2	ビット1をクリアする
CBIT<op>	CBIT dest,src,1,src,2	ビット1をクリアする
IBIT<op>	IBIT dest,src,1,src,2	ビット1をクリアする
TBZ<op>	TBZ dest,src,1,src,2 TBZ dest,src,1,0imm	ビット0をクリアする

【図35】

オペランド	アセンブリ記法	動作
TBNZ<op>	TBNZ dest,src,1,src,2 TBZ dest,src,1,0imm	ビット0が1の場合はクリアする
AND<op>	AND dest,src,1,src,2 AND dest,src,1,0imm	ビット単位AND
ANDN<op>	ANDN dest,src,1,src,2 ANDN dest,src,1,0imm	ビット単位AND-NOT
OR<op>	OR dest,src,1,src,2 OR dest,src,1,0imm	ビット単位OR
XOR<op>	XOR dest,src,1,src,2	ビット単位XOR
XORC<op>	XORC dest,src,1,src,2	符号なしビット単位XOR
SHR<op>	SHR dest,src,1,src,2 SHR dest,src,1,0imm	論理右シフト
SHL<op>	SHL dest,src,1,src,2 SHL dest,src,1,0imm	論理左シフト
SHRA<op>	SHRA dest,src,1,src,2 SHRA dest,src,1,0imm	算術右シフト
SHRC<op>	SHRC dest,src,1,src,2 SHRC dest,src,1,0imm	算術右シフト
ROR<op>	ROR dest,src,1,src,2 ROR dest,src,1,0imm	右回転
ROL<op>	ROL dest,src,1,src,2 ROL dest,src,1,0imm	左回転
INS	INS dest,src,shift_count,shift_count	挿入
EXT	EXT dest,src,shift_count,shift_count	抽出
CNT	CNTLSO dest,src CNTLSZ dest,src CNTLSR dest,src CNTMSO dest,src CNTMSZ dest,src CNTMSR dest,src	下位のビット数をカウントする 下位のビット数をカウントする 下位のビット数をカウントする 上位のビット数をカウントする 上位のビット数をカウントする 上位のビット数をカウントする

【図36】

オペランド	アセンブリ記法	動作
CRCP	CRCP dest,src,1,src,2	指数比較
MECP	MECP dest,src	指数測定
SYM	SYM dest,src,table_type,shift_count	対称テーブルアクセス
ACC	ACC<op> dest,src,1,src,2 ACC<op><src> dest,src,1 ACC<op><src> dest,0imm	累算
ACCN	ACCN<op> dest,src,1,src,2 ACCN<op><src> dest,src,1 ACCN<op><src> dest,0imm	累算
MUL<op><op>	MUL dest,src,1,src,2 MUL dest,src,1,0imm	乗算
MAC<op><op>	MAC dest,src,1,src,2,src,3	乗算加算
MACN<op><op>	MACN dest,src,1,src,2,src,3	乗算減算
UMUL<op><op>	UMUL dest,src,1,src,2	ユニバーサルハードウェア乗算
UMAC<op><op>	UMAC dest,src,1,dest,src,2,src,3	ユニバーサルハードウェア乗算加算
DMUL<op>	DMUL dest,src,1,src,2	2倍乗算
DMULN<op>	DMULN dest,src,1,src,2	負の2倍乗算
DMAC<op>	DMAC dest,src,1,src,2,src,3	2倍乗算加算
DMACN<op>	DMACN dest,src,1,src,2,src,3	負の2倍乗算加算

フロントページの続き

(72)発明者 ケニス・イー・ギャレイ
アメリカ合衆国、92714 カリフォルニア
州、アーバイン、フレンズ・コート、
17531

(72)発明者 ジョージ・エイ・ワトソン
アメリカ合衆国、92635 カリフォルニア
州、フラートン、ツリービュー・ブレイ
ス、2952

(72)発明者 ジョン・アール
アメリカ合衆国、92680 カリフォルニア
州、タスティン、ウィリアムズ・ストリー
ト、15512-ビー